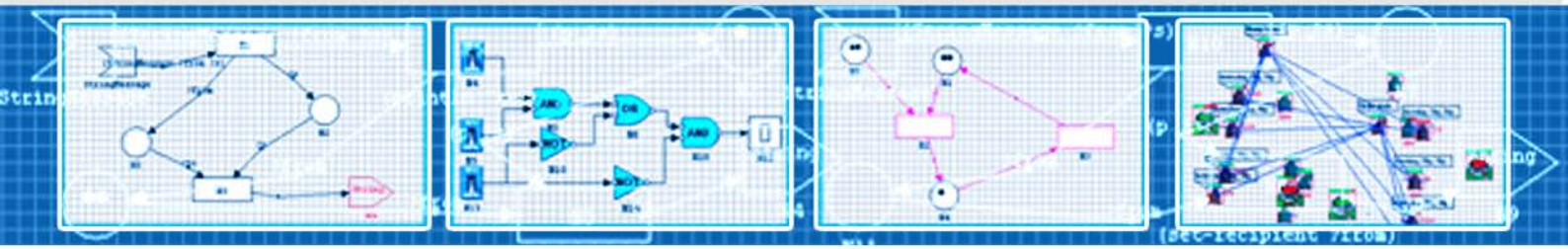

**CIRAD**
**Méthodes Informatiques Modélisation et Simulations Agents (MIMOSA)**

**UCAD-ESP**
**Equipe modélisation, simulation systèmes complexes**

## Editeur de Modèles MIMOSA



# Guide de l'utilisateur

Guide du programmeur de formalisme

Guide du modélisateur

**Auteurs:**
Mohamed Bécaye Touré
Alassane Bah
Grégoire Leclerc

Powered by

# Table des matières

# Chapitre I. Introduction

Depuis une vingtaine d'année, la communauté francophone de recherche en simulation multi agents s'est structurée et distinguée par des travaux originaux et suivi sur la gestion de l'environnement et des territoires. Un réseau de compétences multidisciplinaire s'est ainsi constitué, supporté par un important travail informatique, qui a permis de développer les infrastructures nécessaires. Les équipes ont décidé de fédérer davantage leurs travaux en mettant en place une plateforme commune, la plateforme MIMOSA, en associant le meilleur des expériences et de l'existant technique.

C'est dans le cadre du développement de la plateforme, que l'équipe modélisation, simulation et systèmes complexes (MSSC) de l'université Cheikh Anta Diop de Dakar a été chargé de la conception de l'éditeur de modèle de la plateforme. Pour ce faire, conformément à l'approche collaborative de la plateforme, elle devait lier la plateforme au système SEdit d'édition de modèles de formalismes quelconque. Le système SEdit a été conçu par le LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier).

MIMOSA est une plateforme générique, elle permet une libre définition des formalismes dans lesquels s'expriment les modèles. Cette liberté repose sur le principe que tout formalisme peut être défini à partir de notions de base élémentaires, à partir d'une boite à outils asémantique. A partir de ce principe la plateforme est structuré en quatre niveaux :

- le premier introduit la boite à outils asémantique
- le second accessible par programmation permet de définir les formalismes des modèles à partir du niveau précédent.
- le troisième permet de décrire les catégories d'objets du discours.
- le quatrième permet de construire le modèle en instanciant les catégories. C'est dans ce niveau que l'édition de modèles se fait.

Les trois derniers niveaux définissent la démarche de modélisation de MIMOSA. Après un exposé de cette démarche qui fera l'objet du chapitre II, nous présenterons le système SEdit à travers cette démarche dans le chapitre III.

# Chapitre II. Démarche de modélisation dans la plateforme MIMOSA

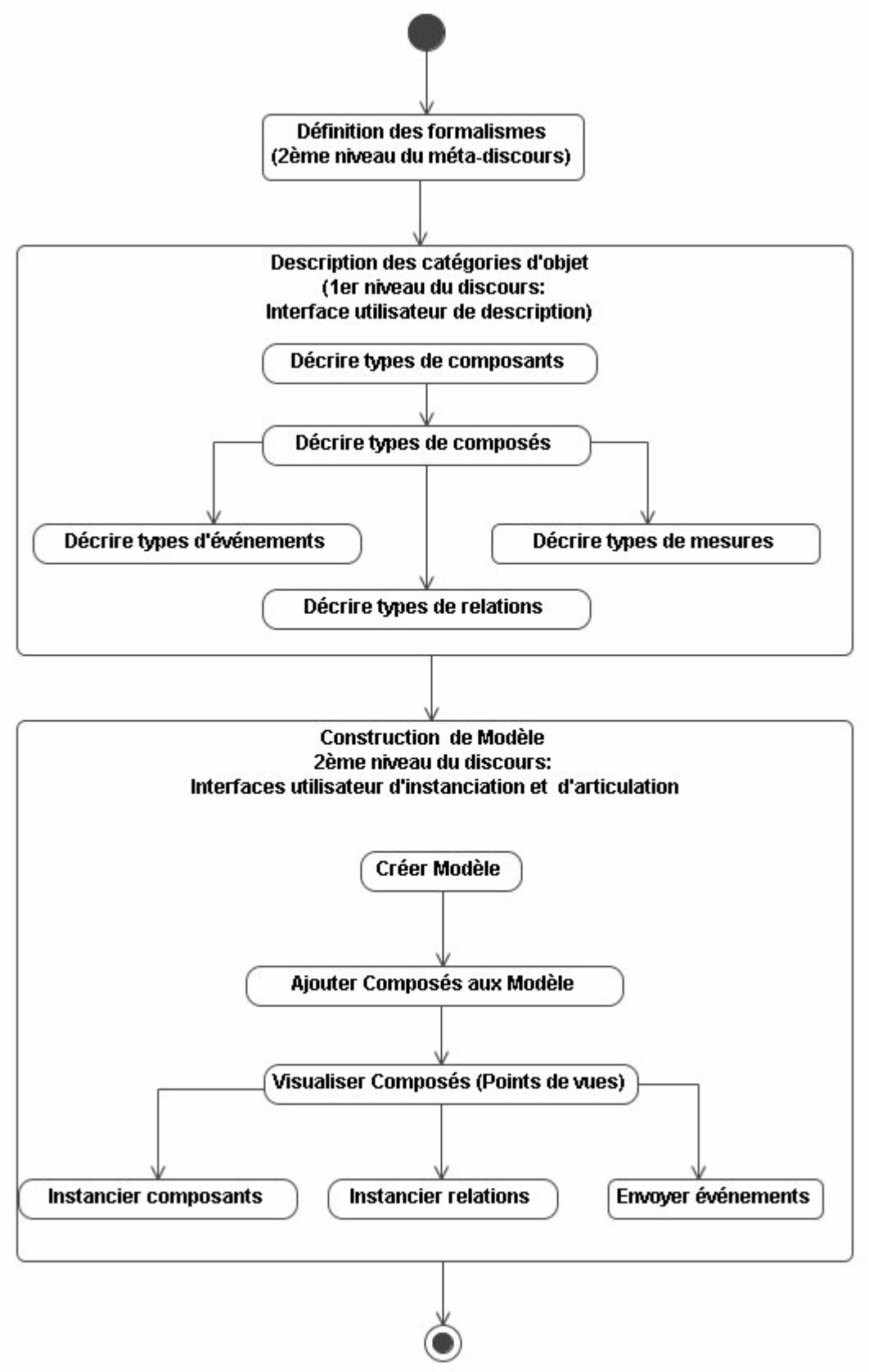
## A. Les étapes de la modélisation

Mimosa est une plateforme de modélisation générique. Elle permet une libre définition des formalismes dans lesquels s'expriment les modèles. Pour ce faire, elle
e met à disposition des notions génériques, une boite à outils asémantique, permettant de décrire les différents formalismes pertinents à chaque modélisation. La définition de formalisme constitue par conséquent une première étape dans la modélisation. Une fois les formalismes définis, le modélisateur a la possibilité de décrire les catégories d'objet dont il veut parler, il s'agit du discours sur les catégories. Dans la dernière étape le modélisateur utilise les types, les catégories qu'il s'est donné dans le niveau précédent pour construire un ou plusieurs modèles.

Nous traiterons donc, dans un premier temps, du mécanisme de définition des formalismes dans la plateforme. Ensuite nous aborderons la description des catégories d'objets. Et nous terminerons par le discours sur les objets, qui consiste en l'instanciation des types du discours sur les catégories.

Les étapes de la modélisation dans MIMOSA sont schématisées dans le diagramme de la page suivante.

## Démarche de modélisation

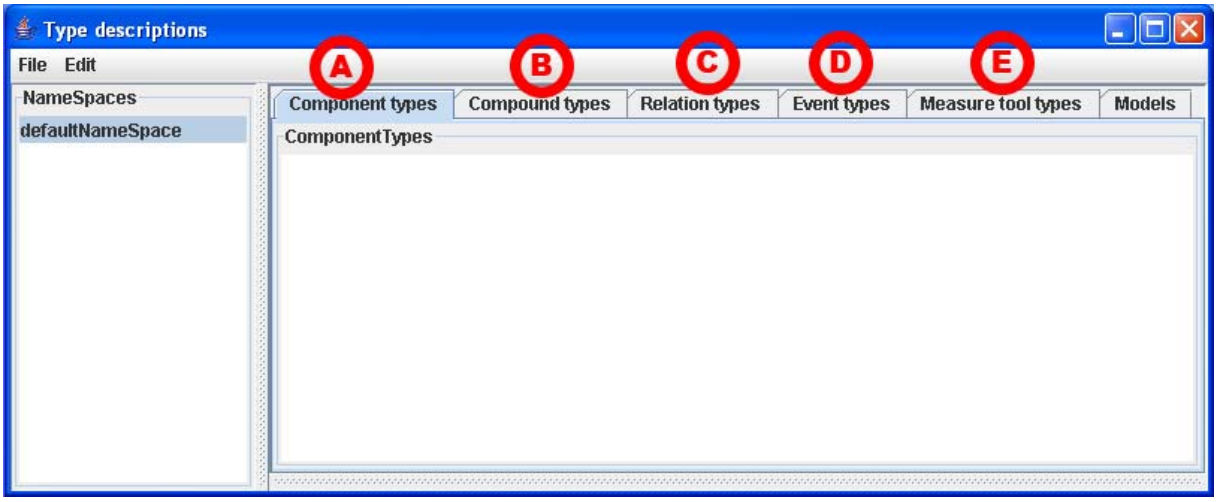


## B. Définition des formalismes (2ème niveau du métadiscours)

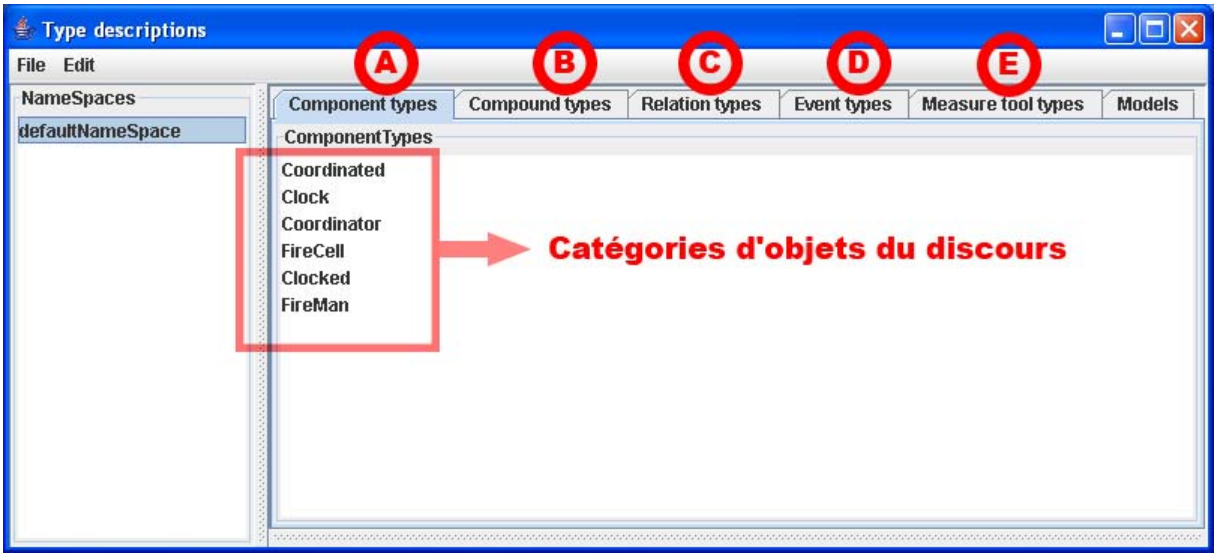
L'ensemble des formalismes sont exprimés à partir de notions de base élémentaires qui de ce fait n'ont pas de sémantique particulière. Ces notions sont divisées en deux catégories :

## C. Description des catégories d'objets (1er niveau du discours)

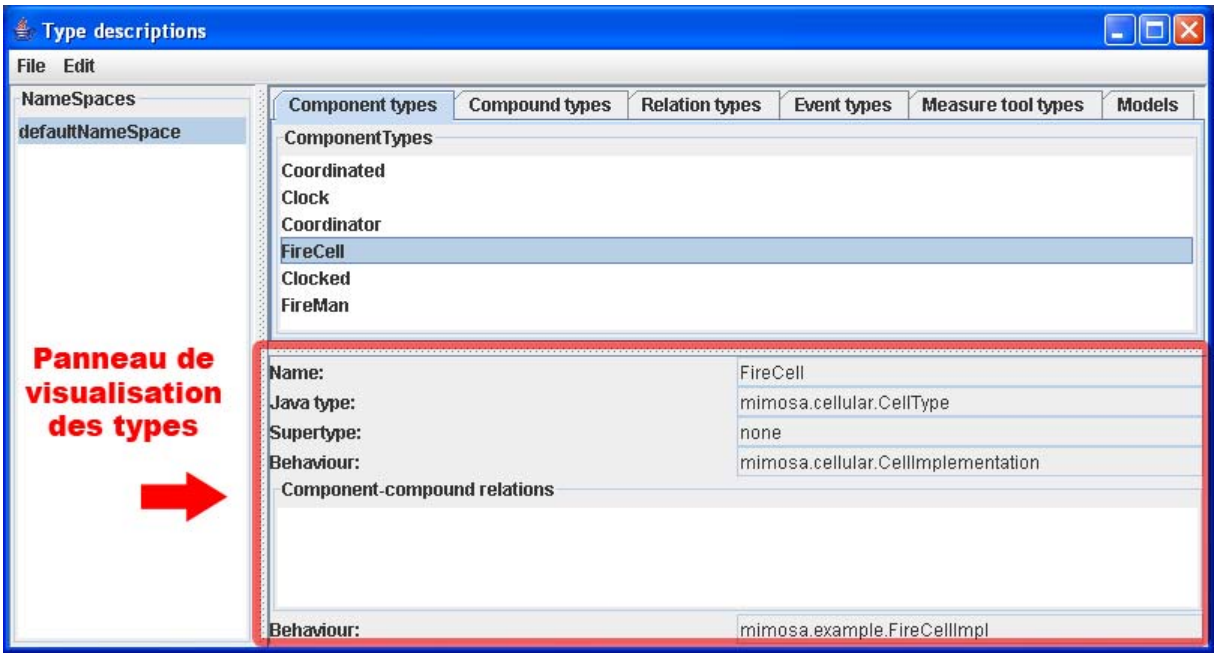
La fenêtre principale de MIMOSA présente plusieurs onglets permettant la description des types d'objets du discours. On retrouvera l'onglet **Component Types** (A), **Compound Types** (B), **Relation Types** (C), **Event types** (D), **Measure tool types** (E). Ces onglets permettent respectivement de définir des types de composants, des types de composés, des types de relations, des types d'événement, des types de mesures.




Chaque onglet est constitué d'une liste des types décrits :

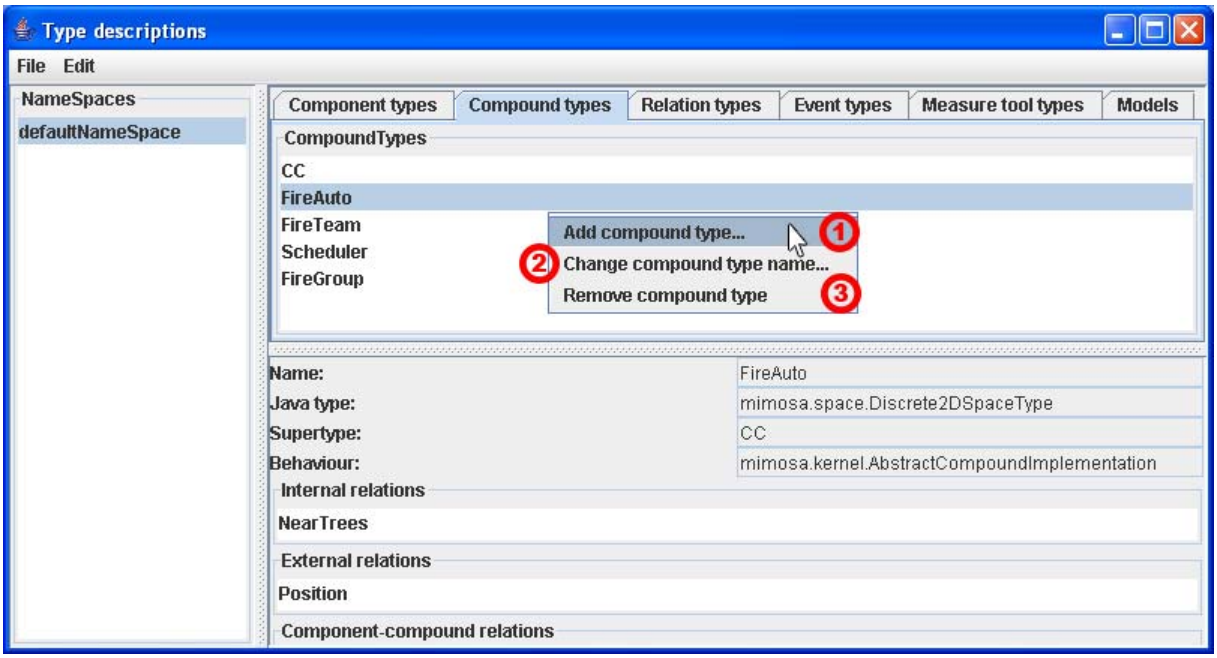


Lorsque vous cliquez sur un type d'objet, sa description apparaît au bas de la liste des catégories.



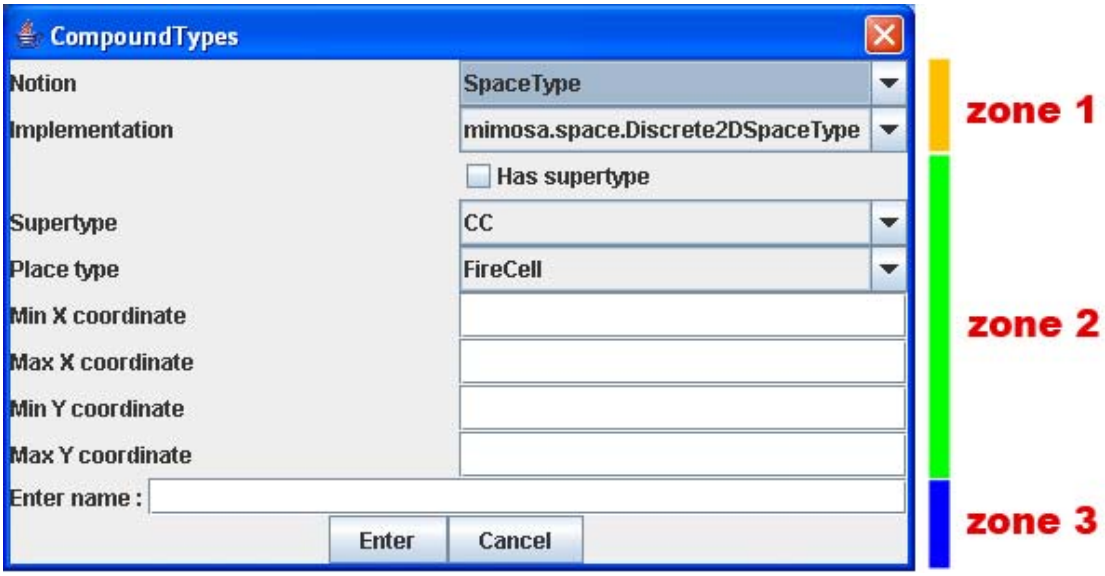

## 1. Opération sur les types

En effectuant un clic droit dans la zone de liste, un popup menu apparaît vous donnant la possibilité d'ajouter un type (**add X type** ①), de changer le nom du type sélectionné (**change X type name** ②), de supprimer le type sélectionné (**remove X type** ③).



## 2. Ajout de types

L'opération d'ajout de type ouvre une boite de dialogue de description de type, après validation, un type, une notion est instanciée.
La boite de dialogue est divisé en trois zones distincte. La première (zone 1 en jaune dans la figure suivante) permet de sélectionner le formalisme auquel appartient la notion que l'on veut décrire. Les formalismes sont catégorisés en **notion,** il faut donc dans un premier temps sélectionner la notion qui correspond à la catégorie d'objet du discours que l'on veut modéliser. Le choix de la notion mets à jour la liste des formalismes, c'est à dire la liste des classes qui implémentent la notion. Une fois la classe d'implémentation sélectionnée le panneau de description du type (zone 2 en vert) est mis à jour. Il faut rappeler qu'un type est une instance de méta-classe implémentant le formalisme d'un concept catégoriel. La description consiste en l'attribution de valeurs aux propriétés de l'instance. Dés lors que le type a été décrit il faut lui attribuer un nom (zone 3 en bleu).



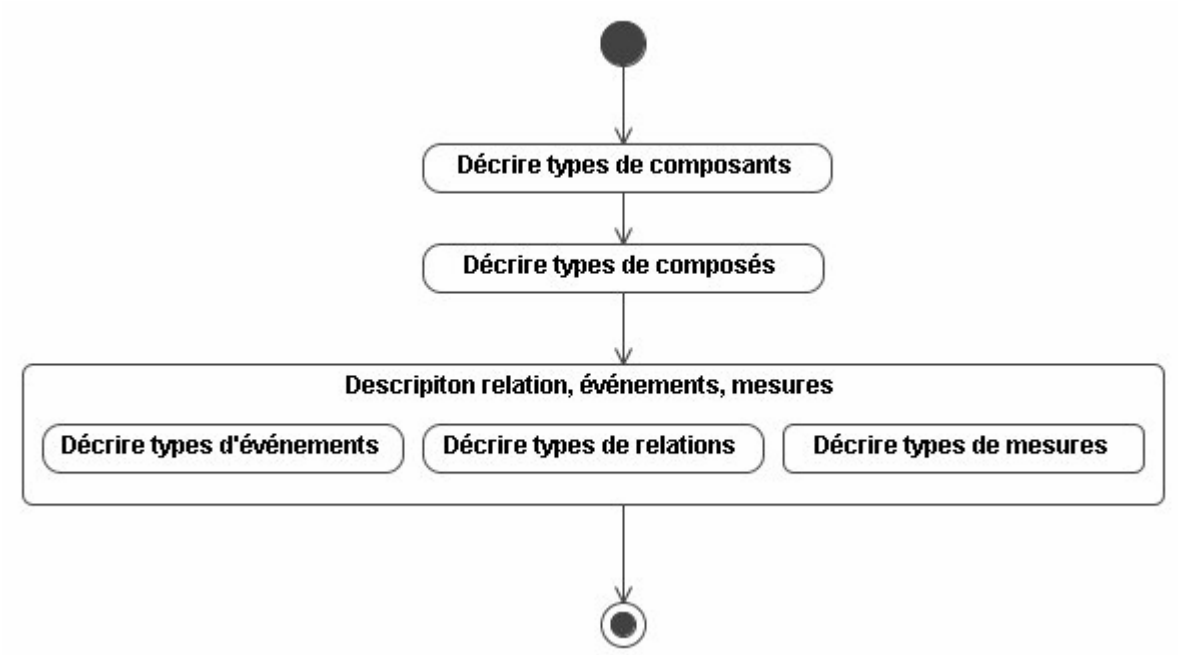
## 3. Séquence d'instanciation des types

Mimosa est multi modèle, multi point de vue. Le composé est la notion de base permettant de décrire un point de décrire un point de vue. Un composé est un agrégat de composants du même type. Décrire un composé revient donc souvent à sélectionner le type de composant du composé. Le type de composant doit donc nécessairement être crée avant le type de composé.

Mimosa permet de décrire trois catégories de relation à savoir la relation entre deux composants d'un même composé (**intra compound relation**), la relation entre deux composés de composés distinctes (**inter compound relation**) et la relation entre un composant et un composé (**component compound relation**). On remarque que la notion de composé est centrale dans la définition d'une relation. Il est donc nécessaire de spécifier à la relation les types de composés sur lesquels elle porte. Les types de relations doivent donc être décris après les types de composés.

Les événements et les mesures portent sur les composants et les composés. Ceux-ci soient décris avant toute description d'événement ou de mesures.

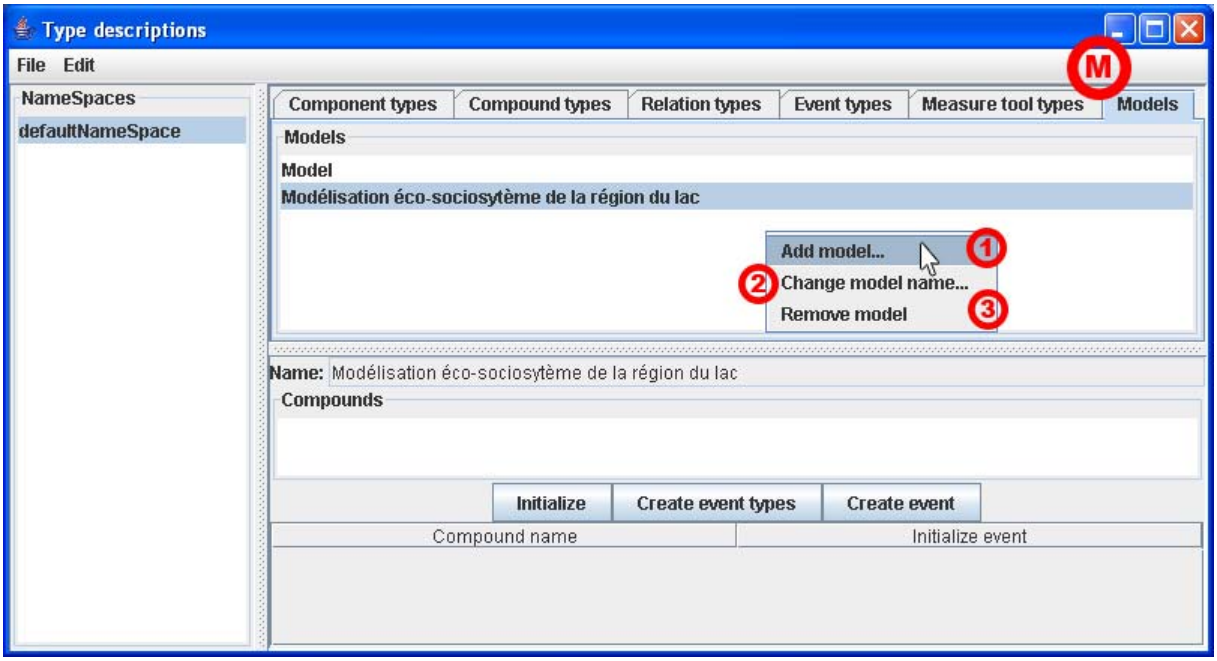
Ces quelques considérations sur la séquence d'instanciation des types sont schématisés dans la figure suivante.



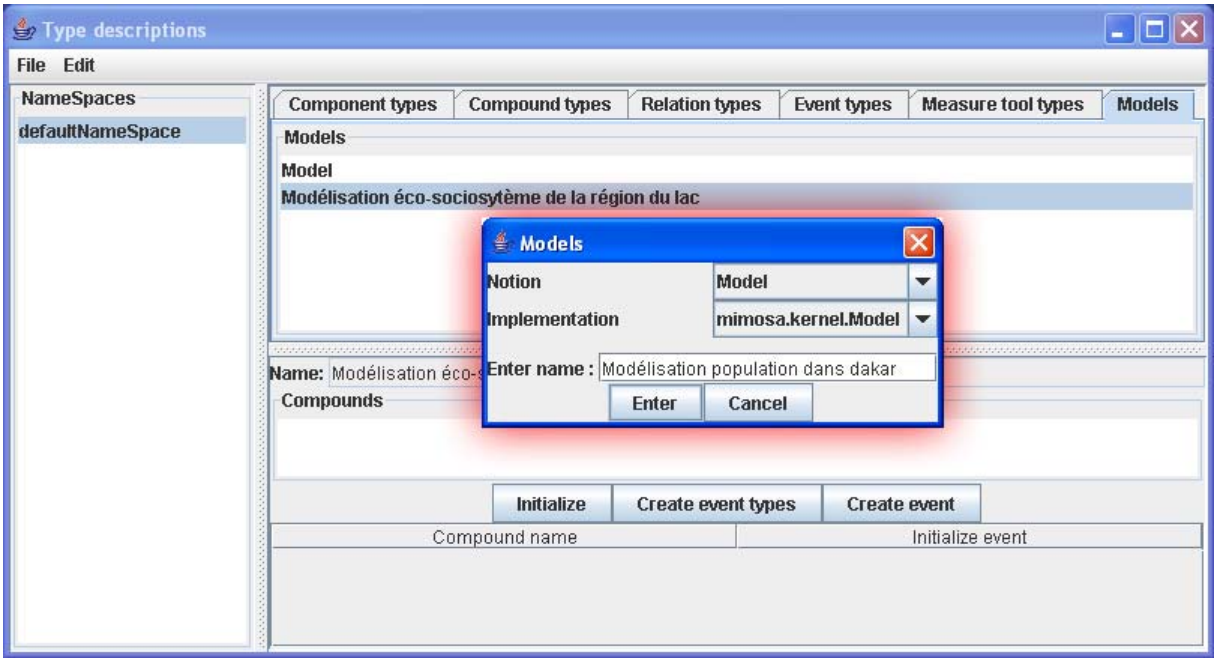
## D. Création de modèles

MIMOSA est un outil de modélisation de systèmes complexes. Le terme modèle dans MIMOSA décrit une articulation des points de vue, une articulation de modèles au sens commun. La gestion des modèles se fait dans l'onglet **Models** (M). L'onglet est notamment constitué de la liste des modèles.


En effectuant un clic droit sur la liste des modèles, il est possible d'ajouter un modèle (**add model**①), de changer le nom du modèle sélectionné (**change model name** ②) , de supprimer le modèle sélectionné (**remove model** ③)



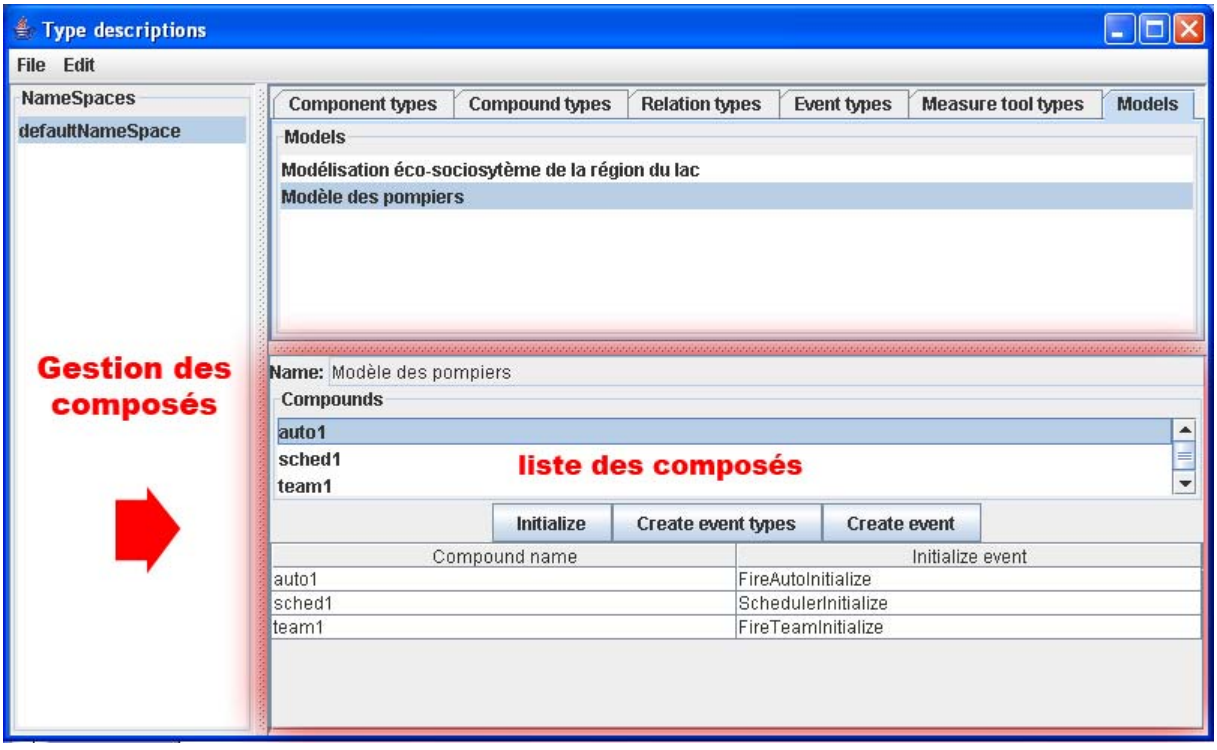
L'action ajout de modèle (**add model**) ouvre la boite de dialogue suivante :



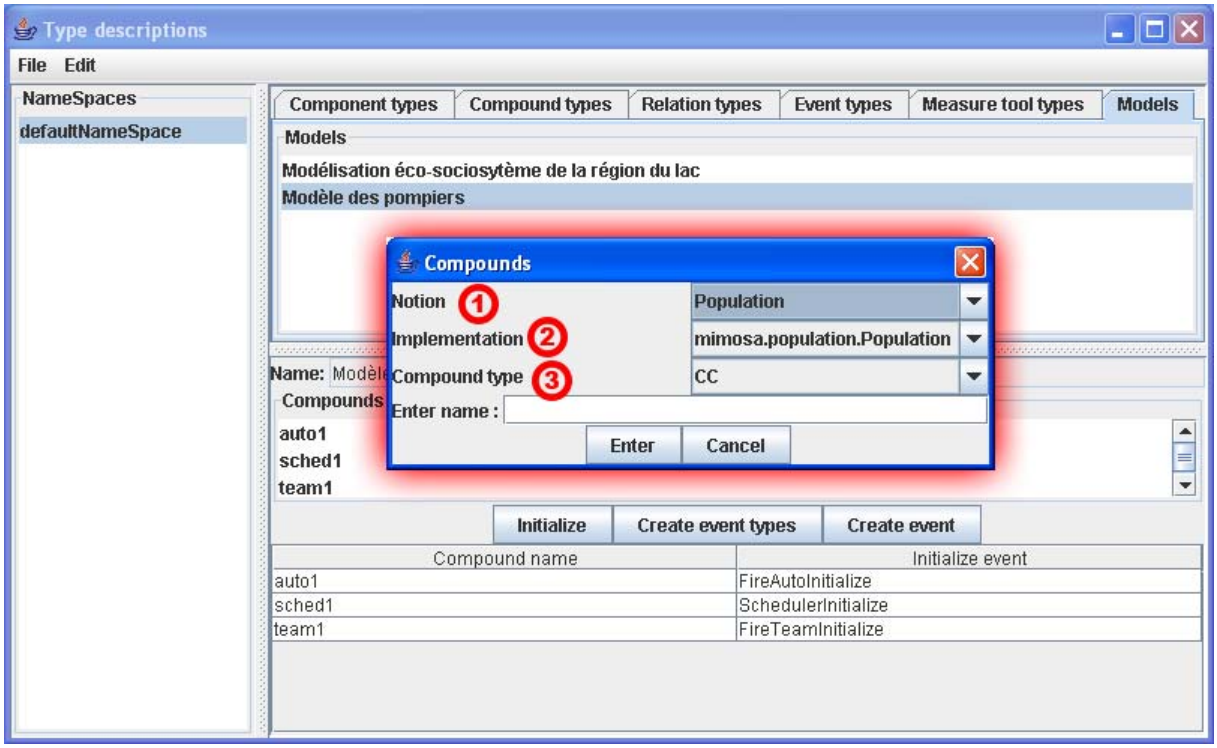

Il suffit de sélectionner une implémention pour la notion de modèle, l'implémentation par défaut est la classe **mimosa.kernel.Model**. Il faut ensuite attribuer un nom au modèle.

## 1. Ajout des points de vue (composés) du modèle

Lorsqu'un modèle est sélectionné, un panneau de description du modèle apparaît (voir figure suivante). Un modèle est rappelons le une articulation de composés ou points de vue. Le panneau de description d'un modèle est donc constitué de la liste des points de vue du modèle (**Compounds –** voir capture ci dessous). A la création du modèle cette liste est vide. Comme pour les types et le modèle, il suffit d'effectuer un clic droit sur la liste pour ajouter un composé, changer le nom du composé sélectionné ou encore supprimer le composé sélectionné.



L'ajout de composé ouvre la  boite de dialogue d'instanciation suivante.




Comme pour l'instanciation des autres notion de MIMOSA, pour instancier un composé il faut sélectionner le formalisme de la notion en sélectionnant la catégorie du formalisme (**A** -- dans la figure ci-dessus) et ensuite en sélectionnant le formalisme (**B** – dans la figure). Un composé est une instance d'un type de composé, il faut donc sélectionner le type du composé à instancier (**C**). Les types de composés ont été définis pendant la description des types, c'est-à-dire pendant le discours sur les catégories (voir section **III**).

## 2. Visualisation des composés

Lorsqu'on double clique sur un composé la fenêtre de visualisation du composé s'ouvre. Par visualisation nous entendons qu'un composé appartient à un formalisme et que ce formalisme n'est pas forcément un formalisme graphique et n'est par conséquent pas visualisable. La visualisation consiste à associer un composé à un formalisme graphique. Par formalisme graphique nous entendons aussi bien les formes géométriques au sens large que les caractères de l'alphabet.

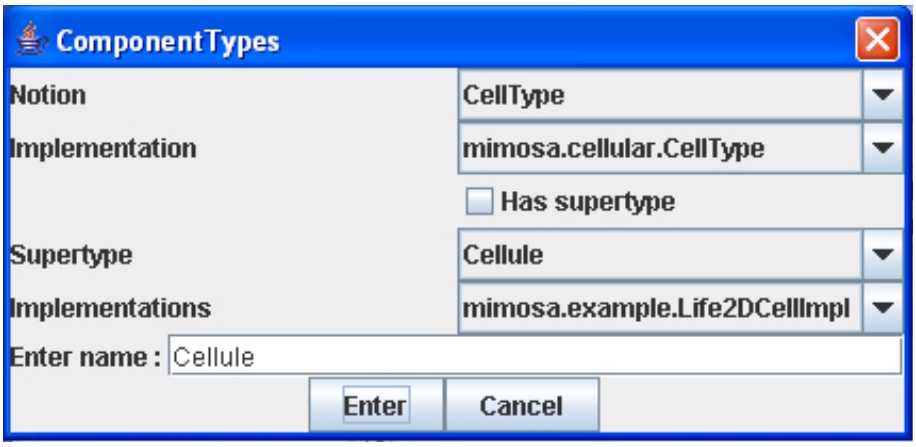
## 3. Instanciations (composants, relations, événements)

Le contenu de la fenêtre de visualisation du composé dépend de son formalisme. A chaque formalisme librement défini il a été attribué une interface personnalisée de définition et de visualisation. Cependant quelque soit l'aspect que peu prendre cette fenêtre elle présente généralement la possibilité d'instancier des composants, des relations et d'envoyer des événements. Il incombe aux éditeurs de formalismes de documenter leurs interfaces. Nous allons néanmoins présenter ces possibilités à travers un exemple mettant en place une population d'agents positionné sur un composé cellulaire.

### a) Définition du composé cellulaire :

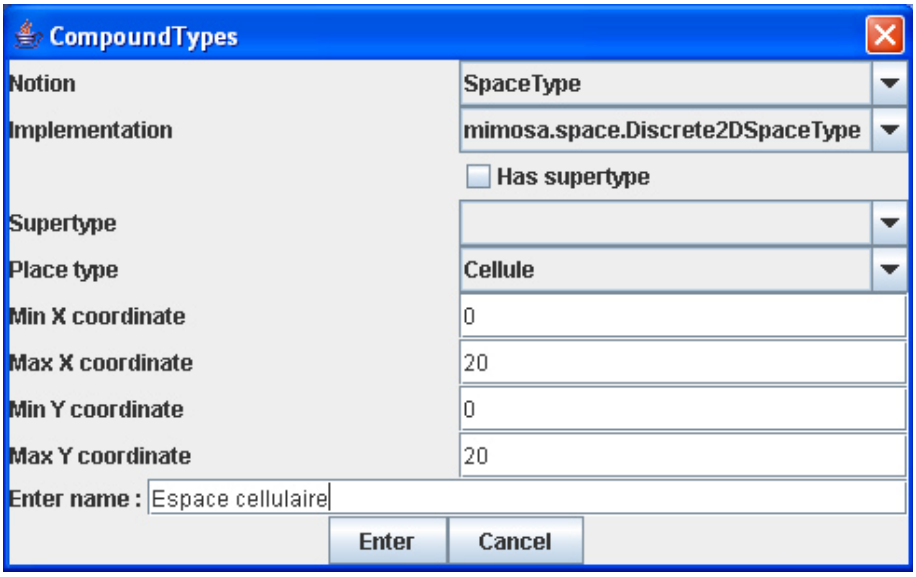
#### (1) Type de composant

| | |
|---|---|
| Notion | CellType |
| Implémentation | mimosa.cellular.CellType |
| Nom | Cellule* |



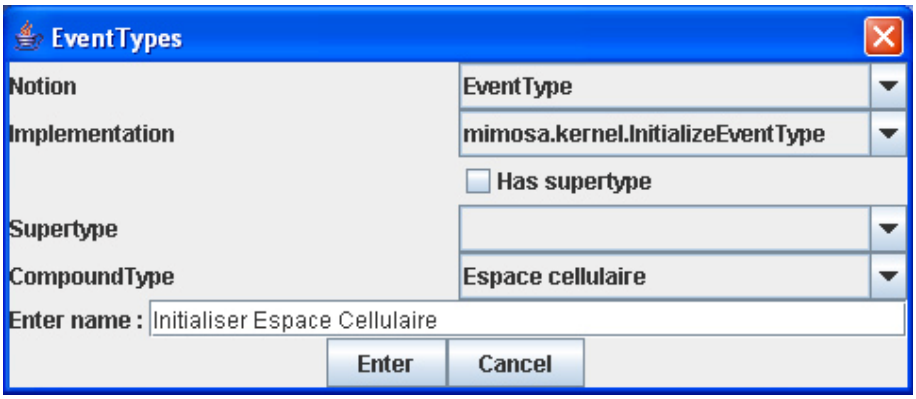

#### (2) Type de composé

| | |
|---|---|
| Notion | SpaceType |
| Implémentation | mimosa.space.Discrete2DSpaceType |
| Type de place | Cellule* |
| Coordonnées minimales en X | 0 |
| Coordonnées maximales en X | 20 (largeur=20 cellules) |
| Coordonnées minimales en Y | 0 |
| Coordonnées maximales en Y | 20 (hauteur=20 cellules) |
| Nom | Espace cellulaire |




(3) Types d'événements

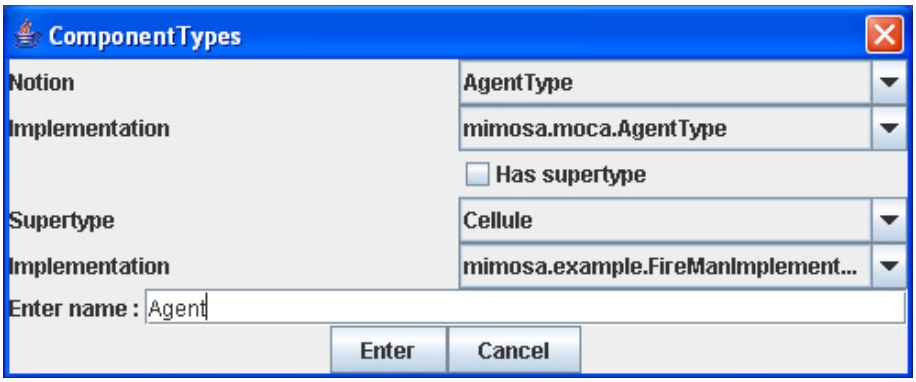
| | |
|---|---|
| Notion | EventType |
| Implémentation | mimosa.kernel.InitializeEventType |
| Type de composé | Espace cellulaire |
| Nom | Initialiser espace Cellulaire |




b) Définition du composé population d'agents

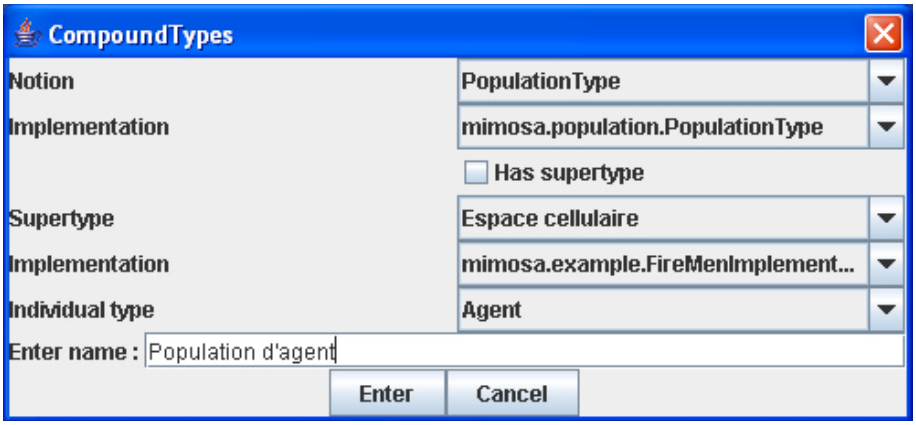
(1) Type de composant

| | |
|---|---|
| Notion | AgentType |
| Implémentation | mimosa.moca.AgentType |
| Nom | Agent* |



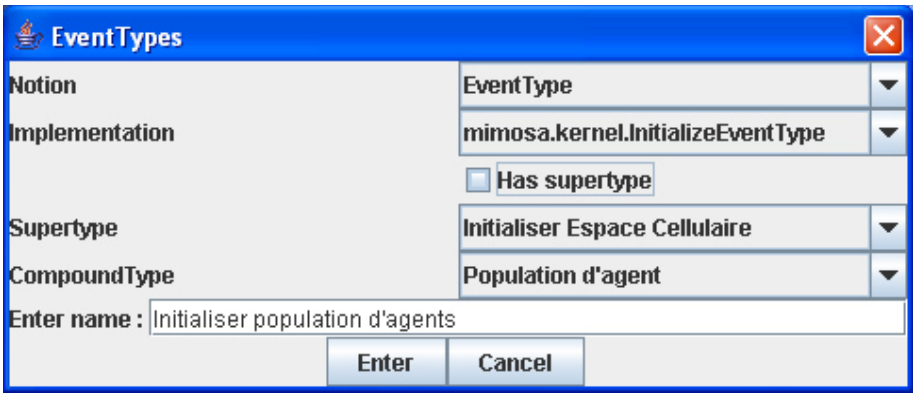

(2) Type de composé

| | |
|---|---|
| Notion | PopulationType |
| Implémentation | mimosa.population.PopulationType |
| Type individuel | Agent* |
| Nom | Population d'agents |



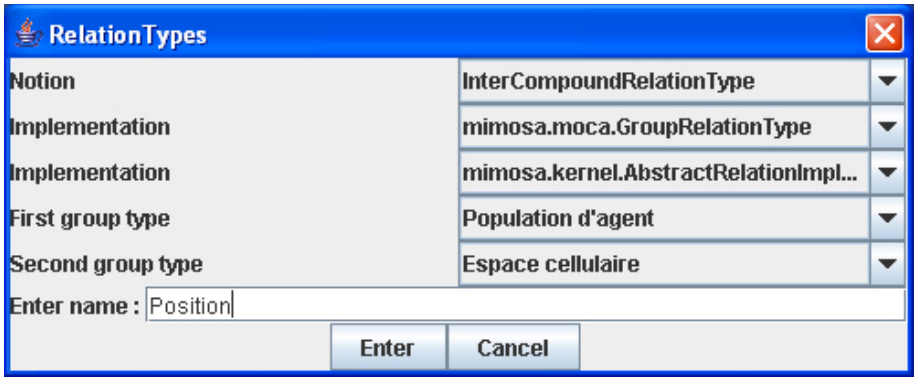

(3) Types d'événements

| | |
|---|---|
| Notion | EventType |
| Implémentation | mimosa.kernel.InitializeEventType |
| Type de composé | Population d'agents |
| Nom | Initialiser population d'agents |



### c) Définition d'une relation de position entre la population d'agents et l'espace cellulaire

| | |
|---|---|
| **Notion** | InterCompoundRelationType |
| **Implémentation** | mimosa.moca.GroupRelationType |
| **Composé 1** | Population d'agents |
| **Composé 2** | Espace cellulaire |
| **Nom** | Position |




### d) Instanciations

Après avoir un ajouté un modèle **Agents dans l'espace**, nous instancions deux composés **espace1** (fig 1) et **population1**(fig 2) instances respectives des types **Espace cellulaire** et **Populations d'agents**.

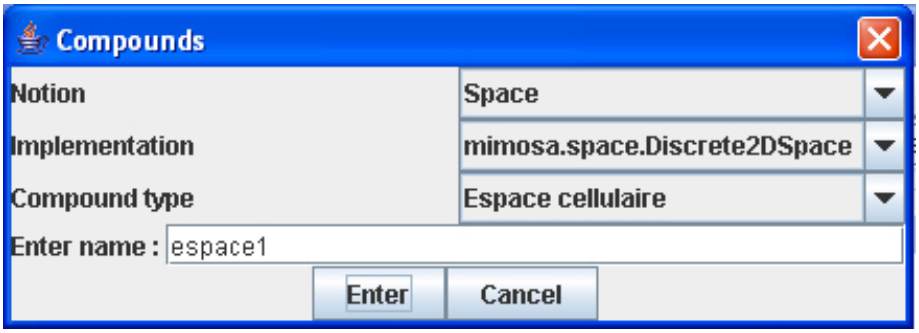



Fig1

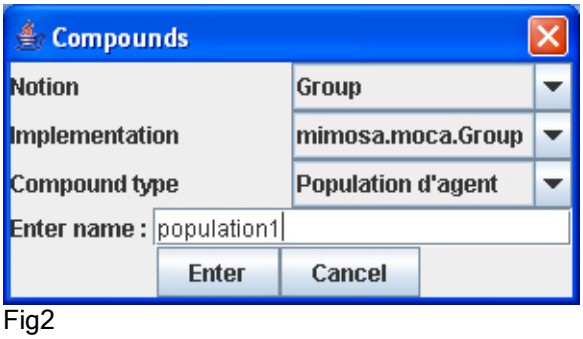
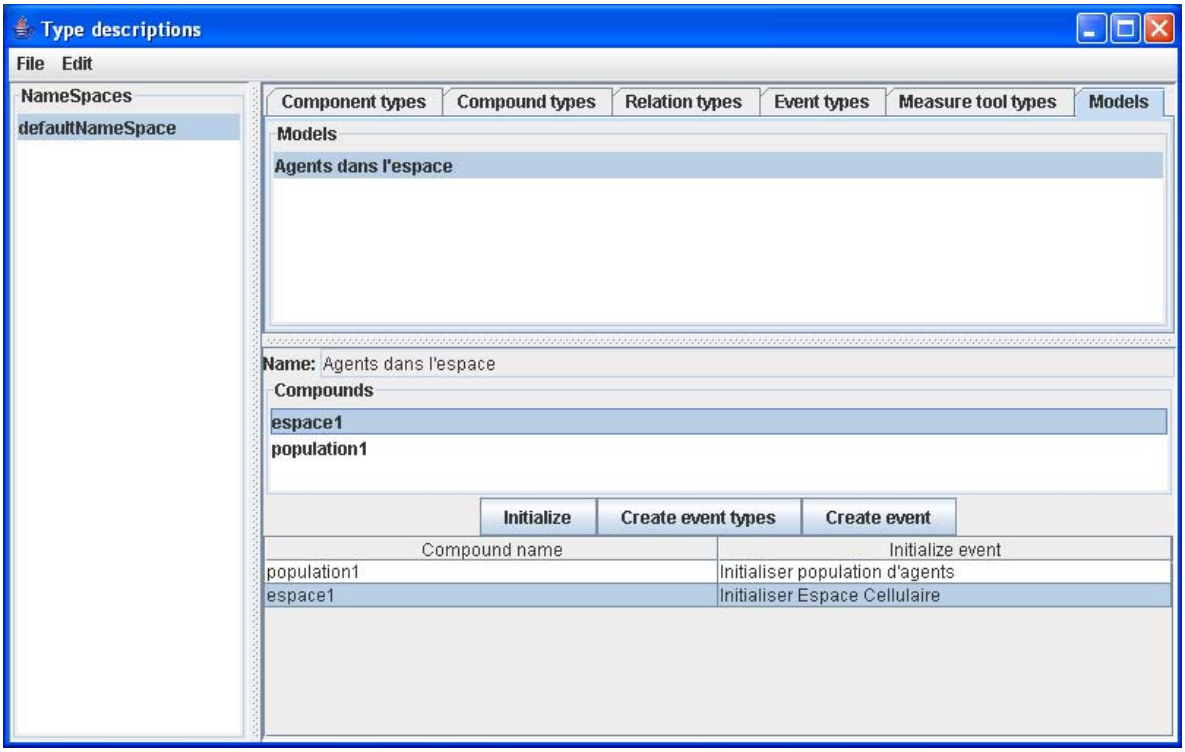
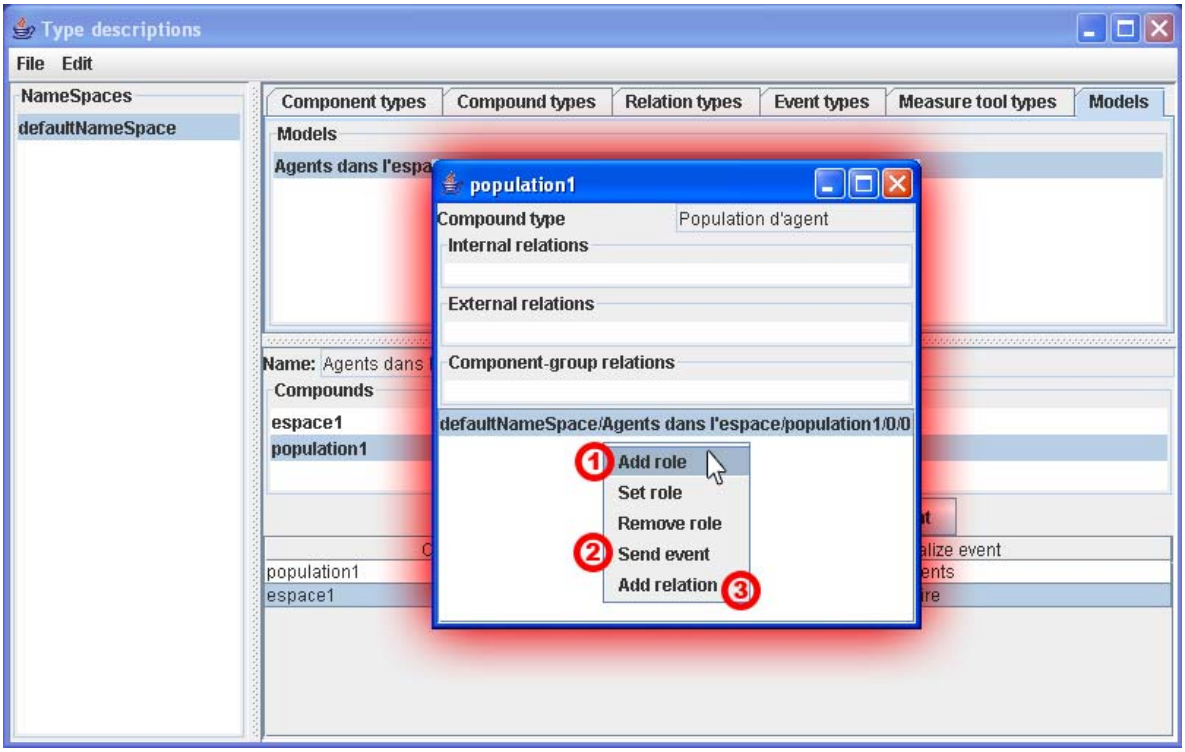



Fig2

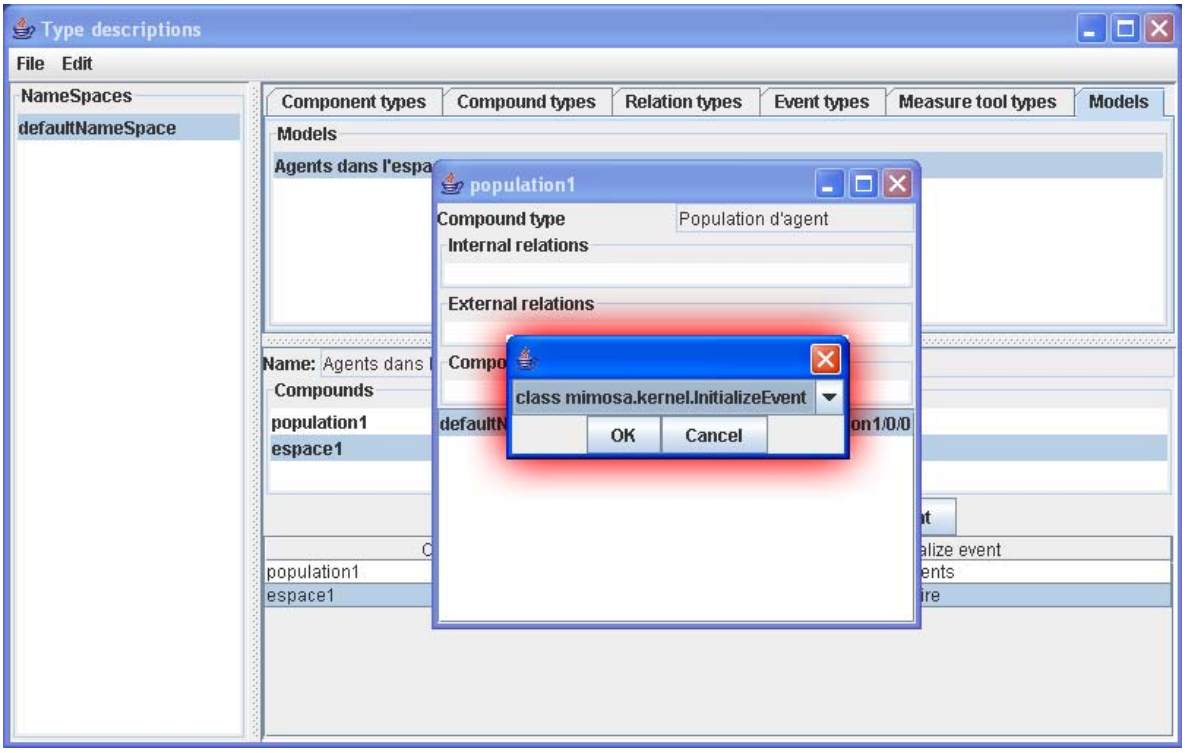
Les composés apparaissent dans la liste des composés :




En double cliquant sur le composé **population1**, sa fenêtre de visualisation apparaît :




Pour ajouter un agent, un composant il suffit d'effectuer un clic droit et de choisir l'option **Add role** ❶. Une fois l'agent sélectionné, l'option **Send event** ❷ permet de lui envoyer un événement (parmi ceux que le concept accepte).




**Add relation** ❸ permet d'ajouter une relation.

# Chapitre III. Le système SEdit

## A. La notion d'éditeur de modèle

D'après l'énoncé du concept de modèle retenu par MIMOSA, un éditeur de modèle est une entité logicielle disposant d'une interface permettant d'instancier des composants, des relations et des composés. L'interface est une interface entre MIMOSA et un humain. L'humain ne peut interagir qu'avec ce qu'il perçoit. Parmi les perceptions de l'humain la vue occupe une place importante. Or les concepts des modèles que l'humain doit articuler ne sont pas forcément exprimés dans des formalismes visualisables. Ce qui implique que pour que l'humain puisse interagir, puisse utiliser les concepts d'un modèle, il est nécessaire pour ces concepts de disposer d'une forme perceptible. En l'occurrence une forme graphique. Par conséquent un éditeur de modèle doit reposer sur un système qui associe aux formalismes du modèle un formalisme perceptible, un formalisme graphique si l'éditeur est graphique.

La généricité d'une plateforme de modélisation repose sur la possibilité de définir librement les formalismes dans lesquels les modèles sont exprimés. Ainsi une plateforme générique doit disposer d'un système de définition des formalismes. Cette description se fait par programmation, le système de définition étant un ensemble de règles et de directives. Lorsqu'on définit un nouveau formalisme, on peut avoir besoin de définir de nouvelles formes graphiques pour représenter les concepts de ce formalisme. On peut avoir besoin de définir un nouveau formalisme graphique. Cela suppose qu'un système de définition de formalisme graphique ait été établi. Cela suppose l'existence d'un mécanisme d'extension des classes de définitions des formalismes graphiques. Et cela constitue la condition indispensable pour l'édition de modèles dans une plateforme générique.

La définition suivante a été retenu : un éditeur graphique de modèle est un outil logiciel permettant d'éditer des modèles par l'intermédiaire de l'édition d'un modèle graphique dont le formalisme est associé à celui du modèle. Cet éditeur doit reposer sur un système d'association et de définition de formalismes graphiques.

Il s'est trouvé que S-Edit l'éditeur de méta modèle développé par le LIRMM, utilisé notamment par la plateforme MADKIT, implémente cette définition d'éditeur générique de modèles. D'où l'idée de coupler ce système à la plateforme MIMOSA.

S-Edit a été développé par Jacques Ferbert et Olivier Gutknecht du LIRMM de Montpellier dans le cadre du projet MADKIT où il est utilisé pour la visualisation et la simulation des agents de la plateforme éponyme. S-Edit est aussi utilisé en version stand alone comme un éditeur générique de modèle.

## B. Les notions du modèle SEdit

S-Edit est un éditeur générique de modèle. Les concepts catégoriels du modèle dans SEdit appartiennent aux formalismes définis par les sous classes de NodeDesc, ArrowDesc, Formalism. Les concepts inviduels du modèle dans SEdit appartiennent aux formalismes définis par les sous classes de SNode, SArrow, Structure.

- **SNode** définit le formalisme de base des concepts individuels unitaires, les composants.
- **SArrow** définit le formalisme de base des concepts individuels de relation entre composant d'un même composé, les relations intra composé.
- **Structure** définit le formalisme de base des concepts individuels composés, les composés.
- **NodeDesc** définit le formalisme de base des concepts catégoriels composants, les types de composants.
- **ArrowDesc** définit le formalisme de base des concepts catégoriels de relation entre composant d'un même composé, les types de relations intra composé.
- **Formalism** définit le formalisme de base des concepts catégoriels composés, les types de composés.

Les concepts du modèle à éditer sont associés aux concepts d'un modèle graphique. Le modèle graphique articule des composants graphiques dont les formalismes sont définis par les sous classes de la classe **GObject**.

S-Edit permet une libre définition des formalismes dans lesquels s'expriment les modèles. Pour ce faire S-Edit en tant que système mets à la disposition des programmeurs de formalismes un mécanisme d'extension des composants, des relations et des composés. L'on notera que le mécanisme de spécialisation des concepts catégoriels n'est pas géré. Ainsi le système est insensible à la spécialisation de **NodeDesc**, **ArrowDesc** et **Formalism**. S-Edit permet par ailleurs une libre définition des formalismes graphiques utilisés pour représenter le modèle. La définition de nouvelles formes graphiques s'effectue par spécialisation de la classe **GObject**.

# C. S-Edit formalisme de MIMOSA

## 1. Vue générale

L'édition de modèle dans S-Edit repose sur une démarche de description des modèles proche de la démarche proposé par Mimosa. Le système S-Edit est présent sur les quatre niveaux de discours définis par MIMOSA :

- Ainsi au niveau de la couche 1 S-Edit mets à disposition la notion de composant appelé nœud, la notion de relation intra composé appelé flèche, la notion de composé appelé structure.
- Dans la couche 2 programmable, plusieurs formalismes sont implémentés tels que les réseaux de pétri, les automates cellulaires, les agents, etc.
- Au niveau de la couche 3 on retrouve la description des catégories d'objets, les types de composants, les types de relations intra composé et les types de composés ; on parlera ici de type de nœud, de type de flèche, de type de structure ou encore formalisme. Cependant à ce niveau on constate que cette description se fait par le biais d'un fichier XML, alors que cela est fait de manière interactive dans MIMOSA.
- Au niveau de la couche 4, l'étape de construction des modèles dans MIMOSA, on retrouve dans S-Edit un éditeur de modèle. Le futur éditeur de modèle MIMOSA.

SEdit étant présent sur tous niveaux du discours et associant de façon native une représentation graphique aux composants et aux relations, il devient envisageable d'utiliser son éditeur de modèle afin d'éditer des modèles MIMOSA. Mais si conceptuellement parlant, MIMOSA et SEdit utilisent les même notions il en n'en n'est évidemment pas de même en ce qui concerne l'implémentation de ces notions. Le tableau suivant listes d'implémentations d'entités conceptuellement identiques.

| Concepts | Implementation MIMOSA | Implementation SEdit |
|---|---|---|
| Concept individuel composant | Component | SNode |
| Concept individuel composé | Compound | Structure |
| Concept individuel relation intra composé | IntraCompoundRelation | SArrow |
| Concept catégoriel composant | ComponentType | NodeDesc |
| Concept catégoriel composé | CompoundType | Formalism |
| Concept catégoriel relation intra composé | IntraCompoundRelationType | ArrowDesc |

SEdit est un système qui associe à un composant une représentation graphique, ce composant est implémenté par la classe SNode au niveau de SEdit et par la classe Component au niveau de MIMOSA. Pour que le composant MIMOSA puisse disposer d'une représentation graphique il doit y avoir assimilation entre la classe Component et la classe SNode. Il doit en être de même avec toutes les classes listés dans le tableau précédent. De sorte à ce qu'une relation intra-composé soit représenté par une flèche et un point de vue Mimosa soit illustré par une fenêtre SEdit. Comment réaliser cette assimilation ?
D'un point de vue technique propre aux langages orientés objet cette assimilation peut se faire par le biais de trois mécanismes ou procédés : l'héritage, l'encapsulation ou la réécriture.

La première méthodes consiste à voir les classes de SEdit comme une spécialisations des classes de la boite à outils de MIMOSA. On disposera alors dans mimosa de composants noeuds et flèches avec une vue graphique de création associée.

La deuxième consiste à encapsuler les éléments de la boite à outils MIMOSA dans des éléments SEdit. De sorte à ce que la création d'un nœud SEdit entraine la création d'un composant MIMOSA.

La troisième suppose une réécriture des classes de la boite à outils MIMOSA par adjonction du code des classes des éléments de base de SEdit.

L'éditeur de modèle implémente principalement la première méthode. Il faut cependant noter que le module implémenté utilise partiellement la réécriture, pour des raisons purement techniques.

## 2. Couplage par héritage

Dans cette approche la boite à outils de MIMOSA est spécialisé en nœuds, liens et structure. Dans la philosophie de la plateforme cette opération s'apparente à une création de formalisme, en l'occurrence un formalisme SEdit. Cependant les concepts de ce formalisme ne constituent au final qu'une autre boite à outils car, asémantique, qui n'aura d'autre particularité que d'être graphique. Pour les puristes cette approche se trouve être la plus élégante car elle fusionne des éléments conceptuellement identiques. Un nœud se trouve être un composant, une flèche est une relation intra composé et la structure, la fenêtre SEdit est un composé. Ainsi les concepts se retrouvent enrichis de méta données permettant leurs affichage et édition graphique. L'héritage par ailleurs, contrairement à la réécriture, n'entraîne pas l'adjonction directe des méta données aux classes MIMOSA. Ces méta données peuvent être considérées comme encombrantes surtout lorsque l'on développe des formalismes qui ne nécessitent pas de représentation graphique.

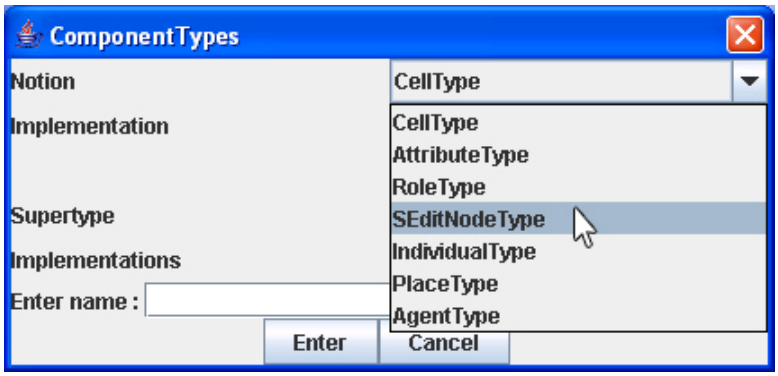
# Chapitre IV. Mise en œuvre

## A. Description des modèles visualisables : les interfaces.

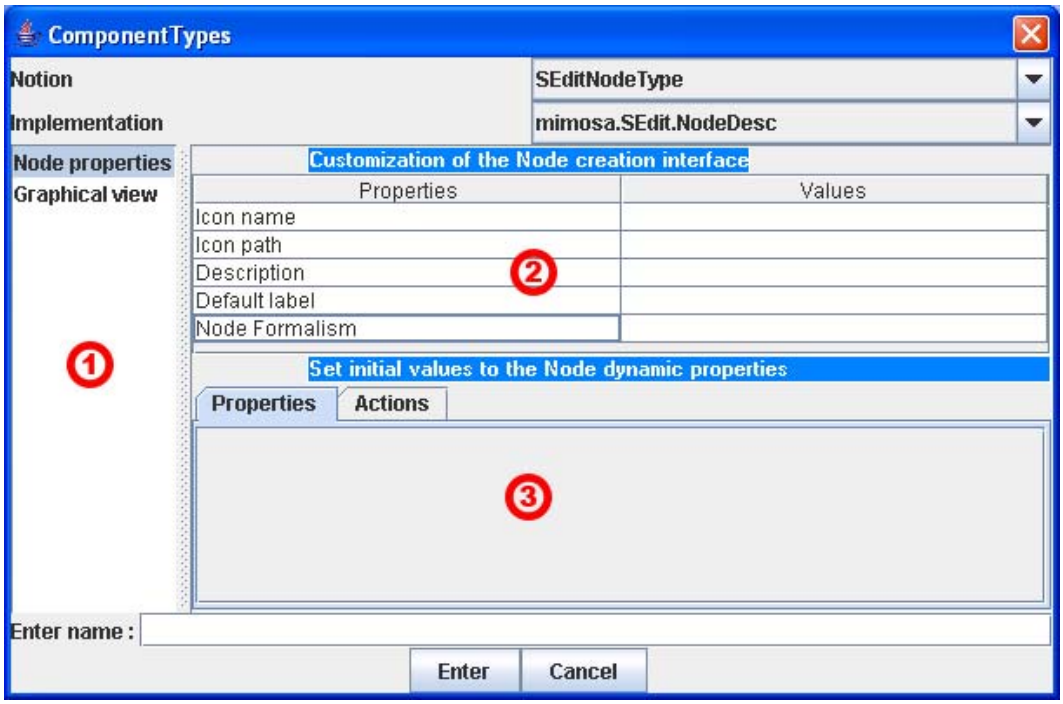
A ce niveau en ce qui concerne le couplage de MIMOSA avec SEdit ; les décisions des éditeurs de formalismes vont légèrement influer sur l'aspect de l'interface graphique de description des types visualisables par SEdit. Ces divergences s'expliquent du fait que SEdit ne dispose à la base que de trois concepts catégoriels : la description de nœuds implémenté par la classe NodeDesc, la description de flèches ou relation implémenté par la classe ArrowDesc et la description des composés représenté par la classe Formalism. Ces concepts catégoriels s'ils ne sont qu'au nombre de trois permettent de catégoriser plus d'une soixantaine de concepts individuels, de composants et de composés. SEdit n'effectue pas une spécialisation systématique des types de composants en fonction de la spécialisation des composants. Cela implique que la description d'un modèle utilisant le méta formalisme SEdit ne se résume qu'à la description du type de nœud, du type de lien, du type de structure.

# 1. Description de noeuds.

L'interface de description de nœuds est accessible à travers la notion **SEditNodeType** d'implémentation **mimosa.SEdit.NodeDesc**




Le panneau suivant de description de nœuds apparaît :




Ce panneau permet d'éditer les propriétés de l'interface graphique concernant le nœud. Il permet de sélectionner le formalisme du nœud ainsi que sa représentation graphique. Il permet par ailleurs d'affecter des valeurs par défaut aux propriétés graphiques ou non du composant. Ces valeurs caractériseront le composant lors de son instanciation pendant la modélisation.

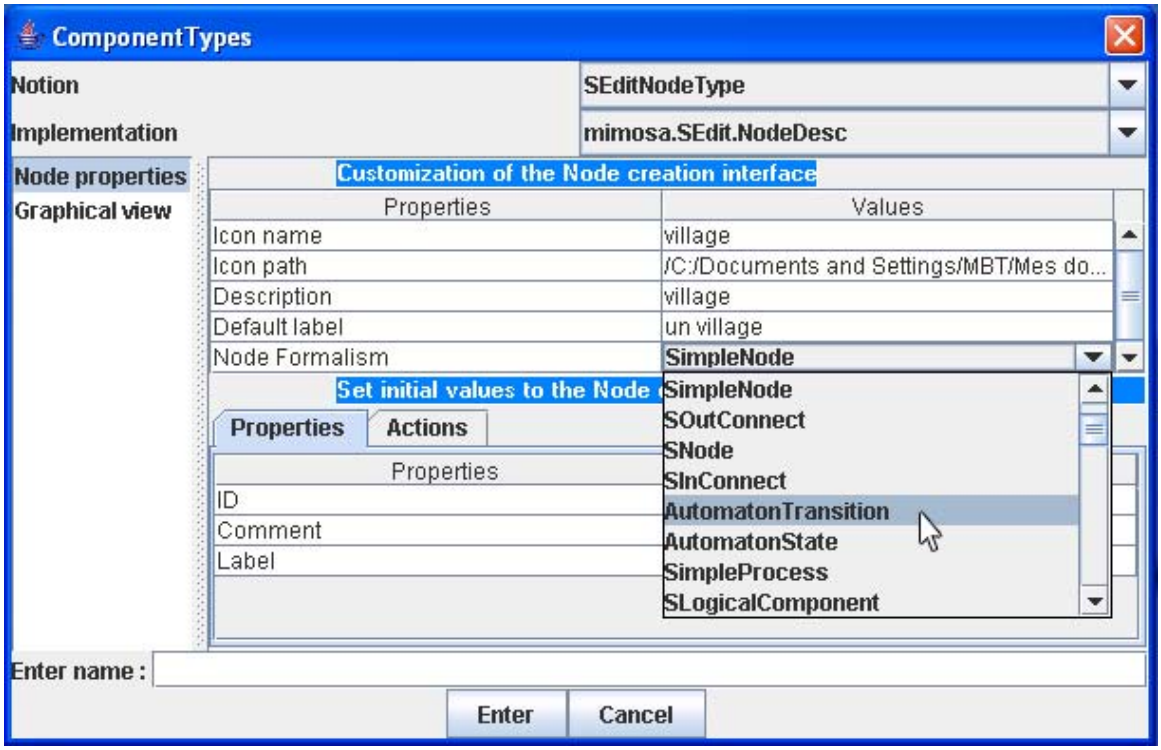
### Définition des propriétés de l'interface graphique

| Customization of the Node creation interface | |
|---|---|
| Properties | Values |
| Icon name | village |
| Icon path | /C:/Documents and Settings/MBT/Mes doc... |
| Description | village |
| Default label | un village |
| Node Formalism | mimosa.SEdit.SimpleNode |

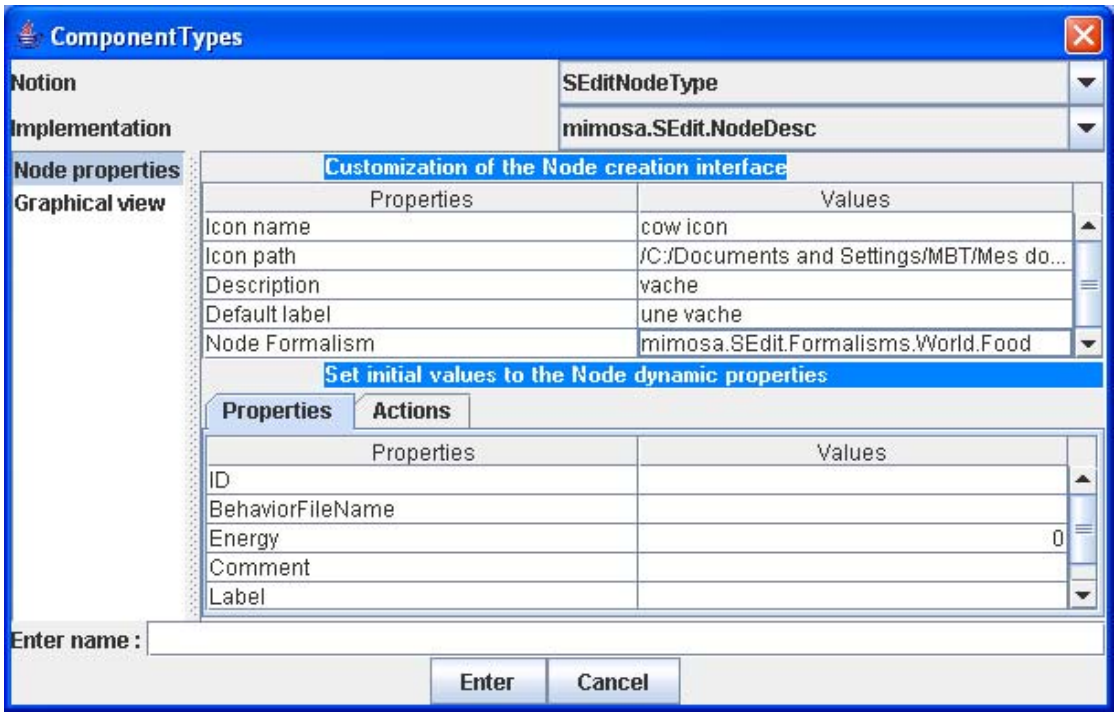
Ce panneau contient les informations nécessaires à la génération d'une barre d'outil propre au model.
Ce panneau permet de paramétrer l'aspect de l'icône (nom, image, description) qui va représenter le composant sur la barre d'outils. Ainsi si nous choisissons une image représentant une vache dans **Icon path**, lors de l'édition des modèles il suffira de cliquer sur la vache pour disposer d'un outil permettant de positionner le composant dans l'écran de S-Edit. Si l'on ne spécifie pas d'image la barre d'outil utilisera le nom du composant.

### Sélection du formalisme auquel appartient le nœud

Sélectionner le formalisme auquel appartient un nœud revient à sélectionner la classe java qui décrit le formalisme. Il s'agit de toutes les classes héritant de **SNode**.

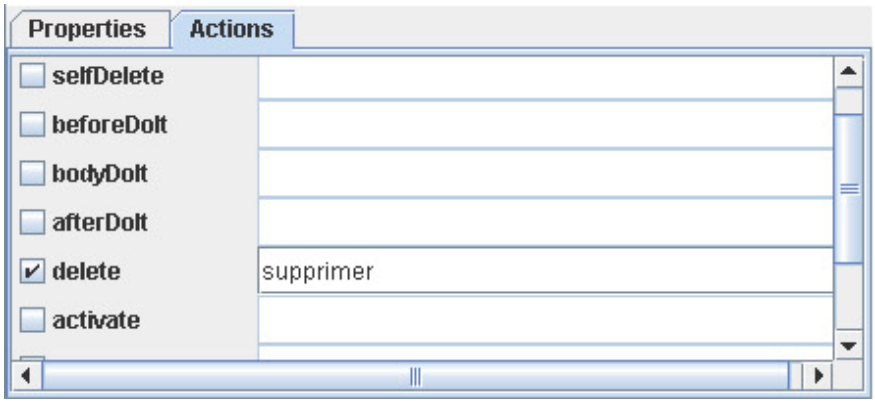


Une fois que le formalisme auquel appartient le nœud a été sélectionné, on remarque que les panneaux sous les onglets de propriétés et actions sont modifiés. En effet, à ce stade on a la possibilité d'attribuer des valeurs initiales aux propriétés du nœud. Les propriétés d'un nœud dépendent du formalisme auquel appartient le nœud.




Il faut noter que les formalismes ayant été développés par plusieurs laboratoires, il est nécessaire de se référer aux guides d'utilisations de ces formalismes afin de connaître le sens de ces propriétés. Pour ce qui est de l'exemple précédent le composant nœud est une entité nourriture qui dispose de propriétés tel que l'énergie. Lors de l'édition des modèles ces propriétés seront accessibles dans le property inspector de SEdit.

De même que les propriétés il est possible de définir les événements ou action que le concept peut recevoir. Pour ce faire il faut sélectionner dans le panneau **Action** une action en lui attribuant un label. Ce label servir à représenter l'action dans le pop menu qui permettra de déclencher celle-ci.

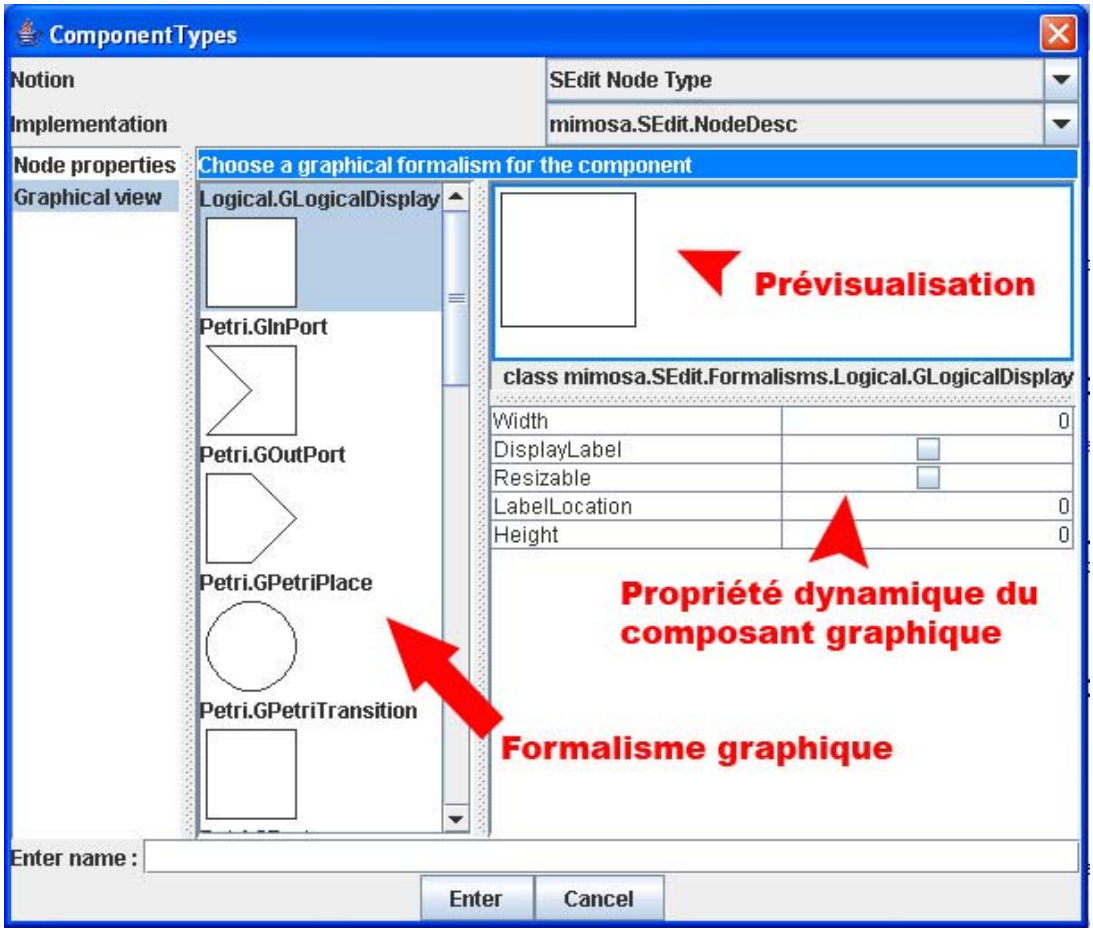



Une action non sélectionné ne pourra pas être accessible pendant la modélisation.

### Représentation graphique du composant

Il faut cliquer sur l'option **Graphic View** du menu gauche de l'écran de description de nœuds.

L'écran de sélection d'un formalisme graphique apparaît. Un formalisme graphique est décrit par une classe héritant de la classe **GObject**.




Dans le menu gauche vous sélectionnez un formalisme graphique. Le concept décrit par ce formalisme apparaît dans le panneau de prévisualisation accompagné d'un property Inspector permettant des fixer des valeurs initiales aux propriétés du composant graphique.

**Remarque :**
La prévisualisation du concept décrit par le formalisme graphique n'est pas toujours disponible car elle peut dépendre des paramètres du composant du modèle qu'il s'agit d'éditer.

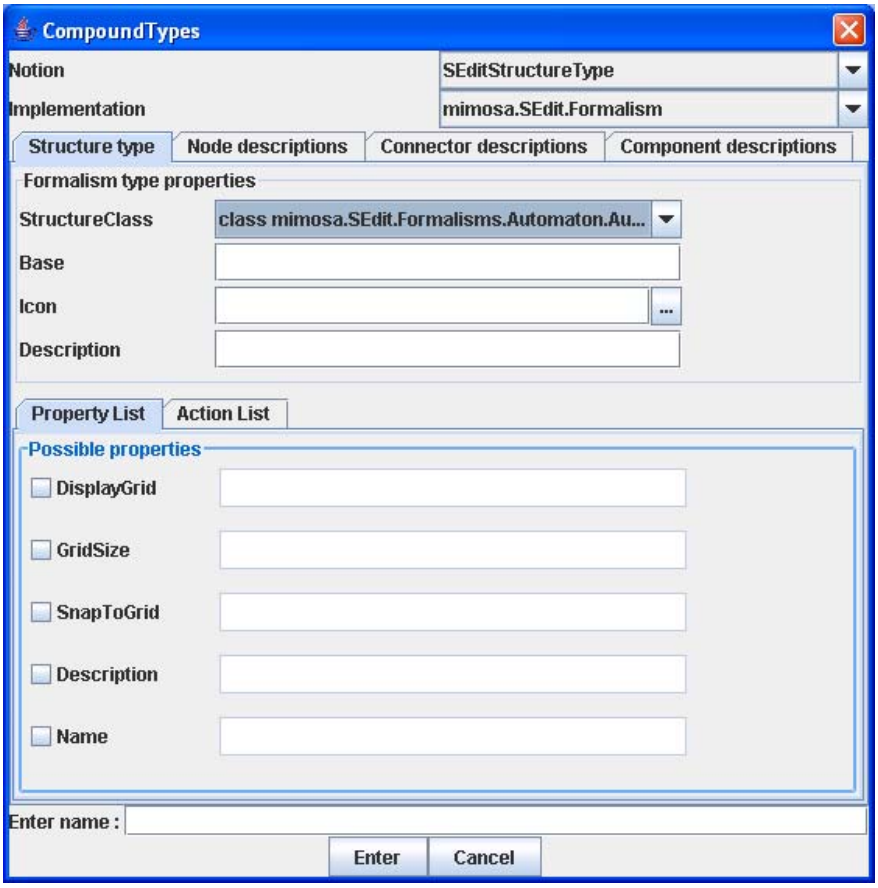
## 2. Description de structure.

Pareillement aux premières étapes placez vous dans l'onglet Compound Types et ajoutez un nouveau type de composé en cliquant **add New Compound Type**. Sélectionnez la notion **SEditStructureType** avec l'implémentation **Formalism .**

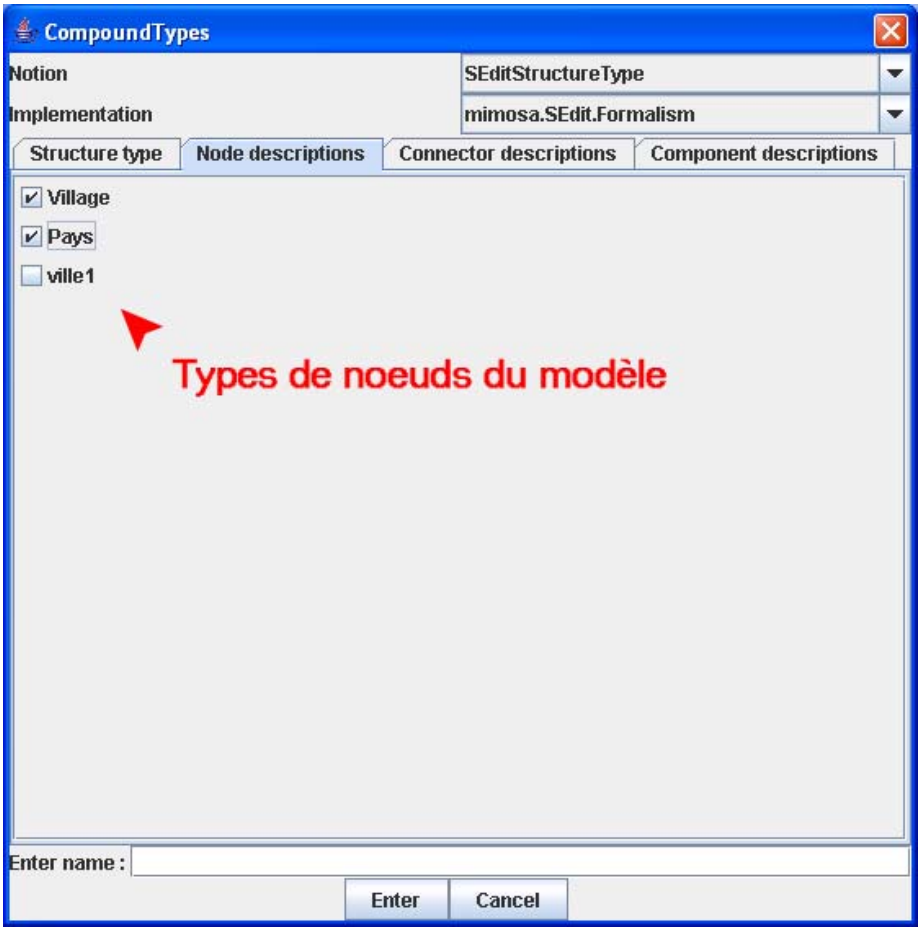
La description de structure permet de paramétrer l'écran dans lequel l'édition de modèles va se dérouler. Cette interface permet de décrire le formalisme général du modèle.

L'onglet SEdit Element Type permet de sélectionnez le formalisme du composé. Le composé n'est rien d'autre que le modèle (au sens du pattern modèle vue contrôleur) de l'éditeur de modèles. Le choix du formalisme se traduit par la possibilité de définir des actions globales sur le modèle ou de définir certaines propriétés de la fenêtre d'exécution propres à certains modèles.

L'attribut **StructureClass** permet de sélectionner le formalisme de la structure

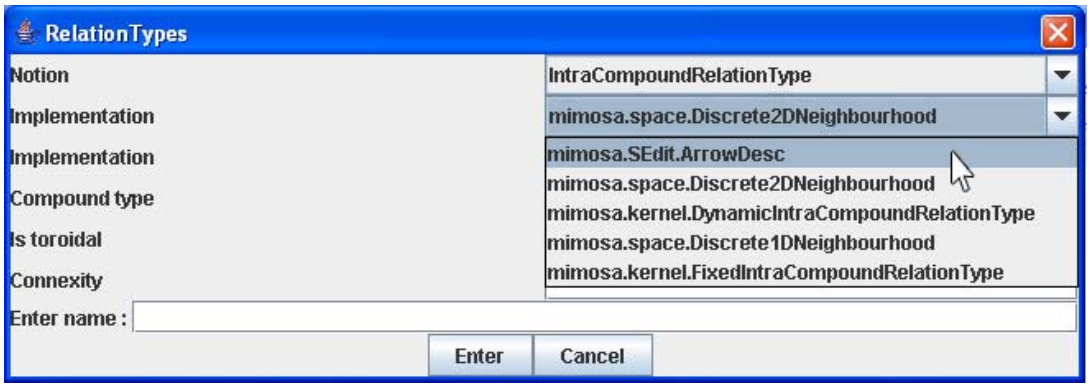


A partir de l'onglet **Node descriptions** on sélectionne les types de composants du modèle. A partir de la liste de noeuds sélectionnés, une barre d'outils est générée permettant d'instancier et de « positionner » les composants.

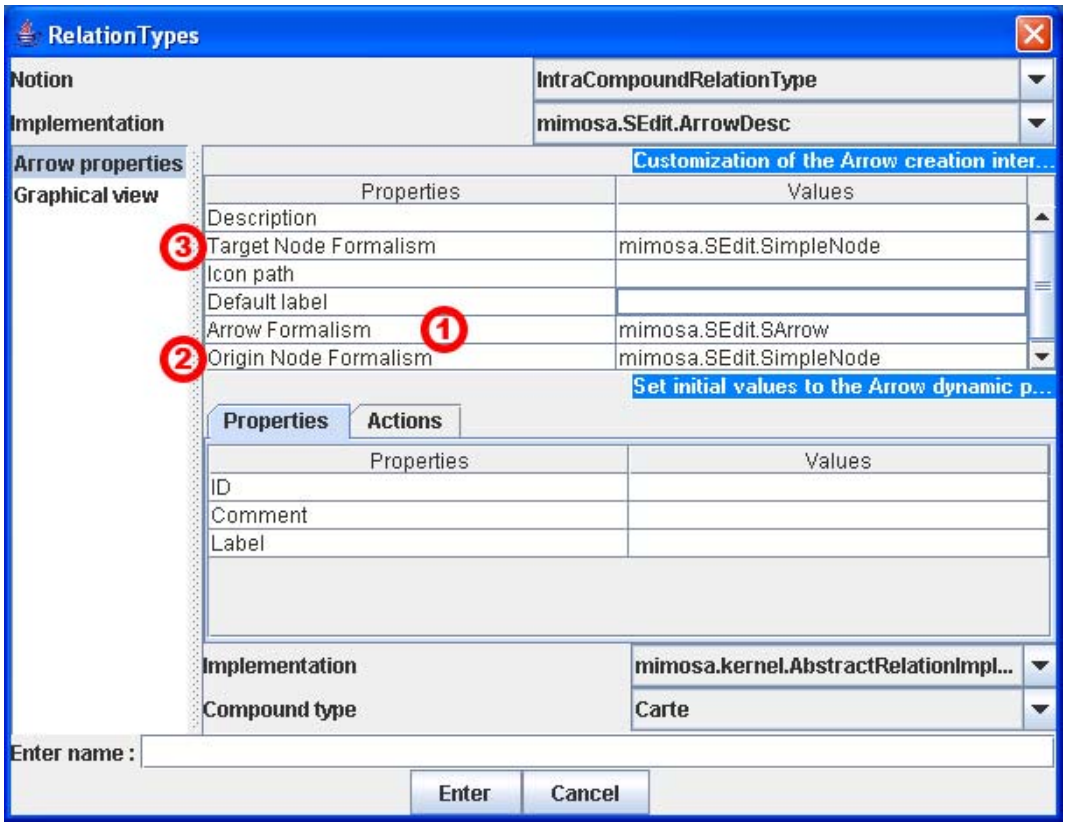


## 3. Description d'arcs.

Sélectionnez la notion **IntraCompoundRelationType**, parmi les implémentations de cette notion vous trouverez l'implementation **ArrowDesc**, sélectionnez là.




Une fois sélectionnée l'écran de description d'arc apparaît :

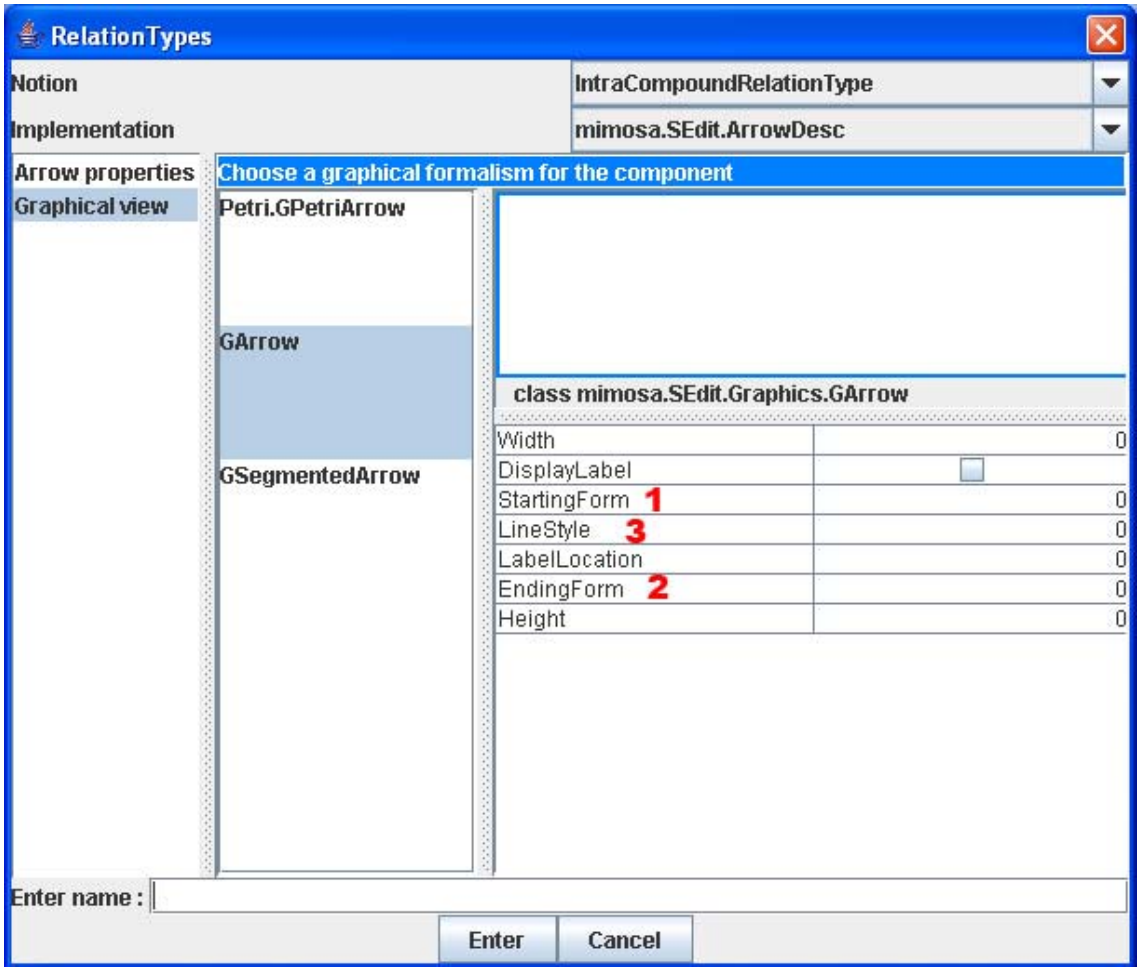



2. La propriété **Arrow Formalism** vous permet de sélectionner le formalisme auquel appartient la relation intra composé.
3. La propriété **Origin Node Formalism** vous permet de sélectionner le formalisme auquel appartient le nœud initial du lien.
4. La propriété **Target Node Formalism** vous permet de sélectionner le formalisme auquel appartient le nœud terminal du lien.

Comme avec les nœuds il est possible d'attribuer des valeurs initiales aux propriétés de la relation. Ainsi que de définir les actions que la relation peut recevoir.

**Représentation graphique d'une relation.**

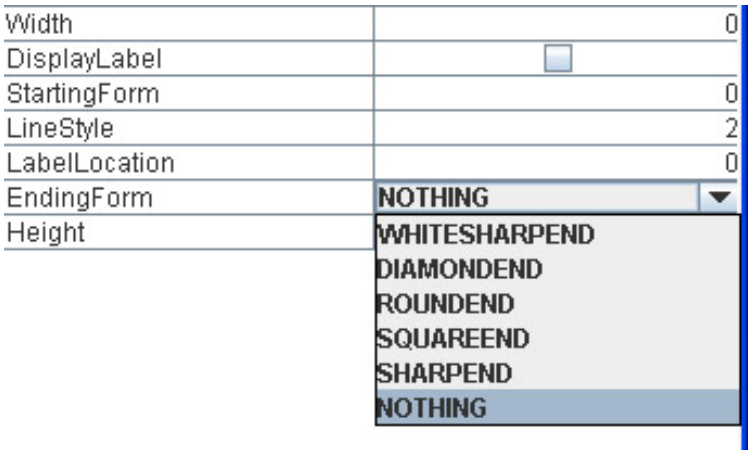
De manière identique aux nœuds, vous pouvez choisir le formalisme graphique utilisé pour représenter la relation :



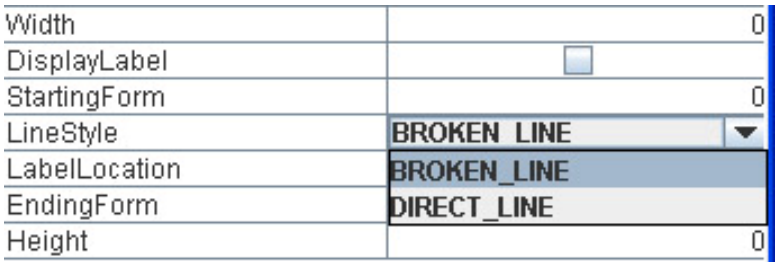

Le formalisme graphique de base pour représenter un nœud est défini par la classe **GArrow**. Les propriétés suivantes sont utiles :

1. **Starting Form** : définit la forme de départ d'un arc, par défaut il s'agit d'une flèche simple: **NOTHING**

2. **Ending Form** : définit la forme de la pointe de l'arc.

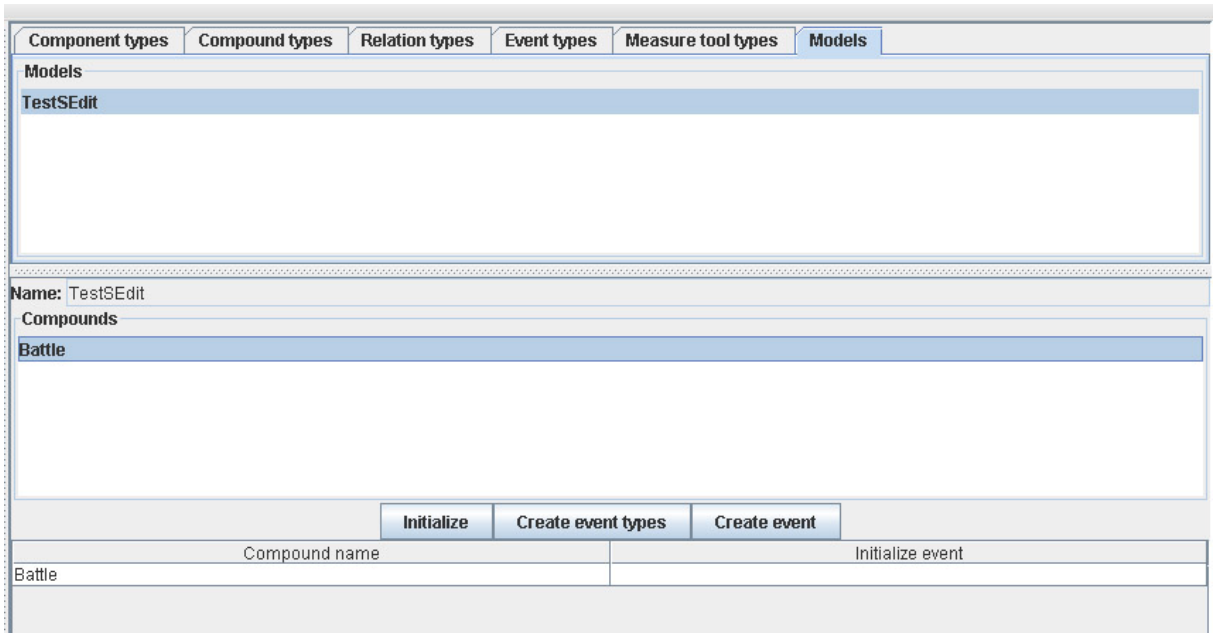



3. **Line Style** : définit si une ligne brisée est utilisé ou une ligne pleine.

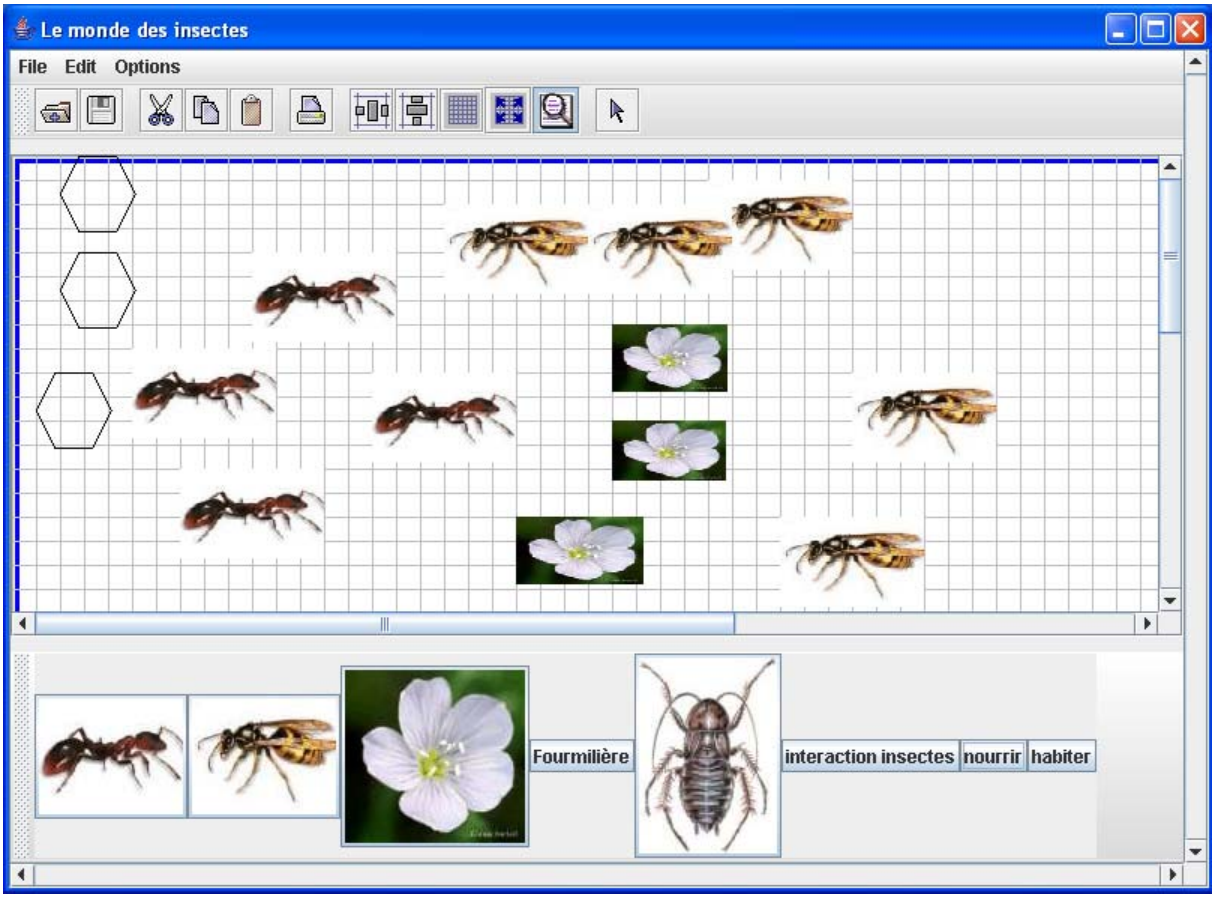


## B. Edition des modèles.

Après avoir constitué un type de composé, allez sur l'onglet Model : créez un modèle. Ajoutez y un point de vue, un composé utilisant le formalisme décrit dans l'interface compoundTypes.




Double cliquez sur le nom du composé (ex battle) la fenêtre de SEdit apparaît avec les composants que vous aviez définis dans sa boite à outils.



# Chapitre V. Description et édition d'un modèle du formalisme des réseaux de pétri.

## A. Le formalisme des réseaux de pétri.

### Les concepts de base

#### 1. La place :

Une condition est la description de l'état d'une ressource du système modélisé :

- une machine est au repos,
- une machine est en réparation
- une commande est en attente

Une condition est soit vraie, soit fausse.

Dans le formalisme des Réseaux de Pétri la condition est modélisée à l'aide d'une PLACE

#### 2. La transition :

Une transition ou un événement est une action qui se déroule au sein du système et dont la réalisation dépend de l'état du système :

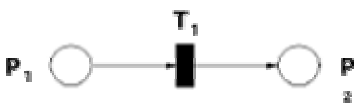
- début de traitement sur une machine,
- panne sur une machine
- début de traitement d'une commande

#### 3. Le réseau de Pétri (RdP)

Un Réseau de Pétri est un graphe orienté comportant :

a. un ensemble fini de places, P= {$P_1$, $P_2$, $P_3$, ..., $P_m$}, symbolisées par des cercles et représentant des **conditions** :
b. une ressource du système (ex. : une machine, un stock, un convoyeur, …)
c. l'état d'une ressource du système (ex. : machine libre, stock vide, convoyeur en panne, …)
d. un ensemble fini de transitions, T= {$T_1$, $T_2$, $T_3$, ..., $T_n$}, symbolisées par des tirets et représentant l'ensemble des **événements** (les actions se déroulant dans le système) dont l'occurrence provoque la modification de l'état du système,


e. un ensemble fini d'arcs orientés qui assurent la liaison d'une place vers une transition ou d'une transition vers une place,

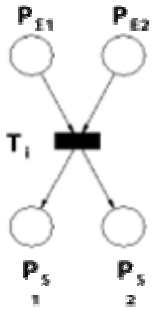


## Marquage d'un réseau de Pétri :

Le marquage d'un RdP est précisé par la présence à l'intérieur des places d'un nombre fini (positif ou nul), de marques ou de jetons. Une place est donc vide ou marquée. Lorsque la place représente une condition logique (ex. : machine à l'arrêt, convoyeur en panne), la présence d'un jeton indique que cette condition est vraie ; fausse dans le cas contraire. Lorsque la place représente une ressource (au sens le plus large – ex. : une machine, un stock), elle peut contenir plusieurs jetons (ex. : nombre de pièces en stock).


## Place d'entrée et place de sortie :

A chaque transition peut être reliée à des places d'entrée et des places de sortie. Les places d'entrée sont les places d'où sont issus les arcs orientés vers la transition ; les places de sortie sont les places où aboutissent les arc orientés issus de la transition.

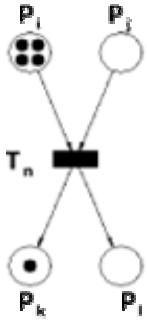



### Remarque :

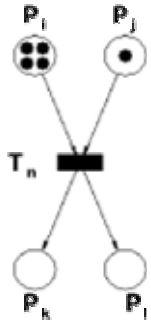
On défini de manière symétrique l'ensemble des transitions d'entrée et l'ensemble des transitions de sortie d'une place.

### Validation d'une transition

Une transition est validée (ou sensibilisée ou franchissable ou tirable) lorsque toutes ses places d'entrée contiennent au moins un jeton.



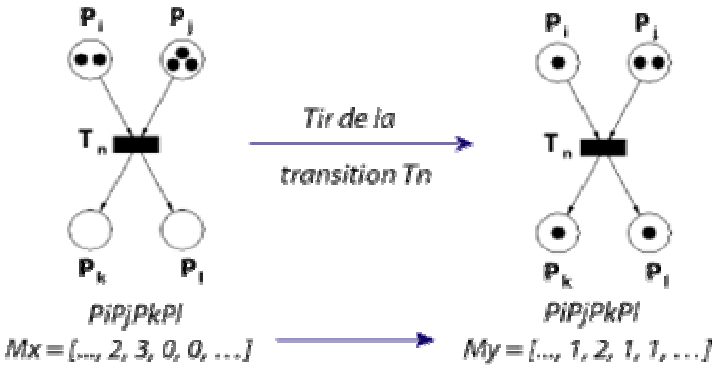

La transition $T_n$ n'est pas franchissable




La transition $T_n$ est franchissable

### Franchissement d'une transition

Le franchissement d'une transition ou tir d'une transition, consiste à retirer une marque dans chacune des places d'entrée de la transition et à ajouter une marque dans chacune des places de sortie de la même transition.

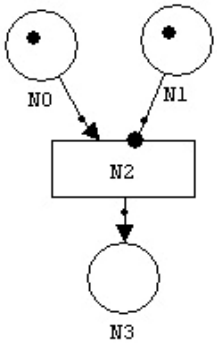



## L'arc Inhibiteur

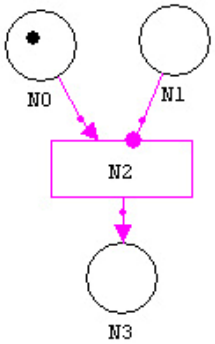
Dans un RdP ordinaire une transition est validée lorsque chacune de ses places d'entrée contient au moins un jeton. Cette règle ne permet pas de réaliser un test à zéro ou autrement dit de conditionner le franchissement d'une transition à l'état vide de l'une de ses places d'entrée.

Les RdP à Arcs Inhibiteurs, permettent de réaliser ce test à zéro.

Dans le RdP ci-dessous l'arc inhibiteur est représenté par une flèche avec un cercle à l'extrémité.




Transition invalide

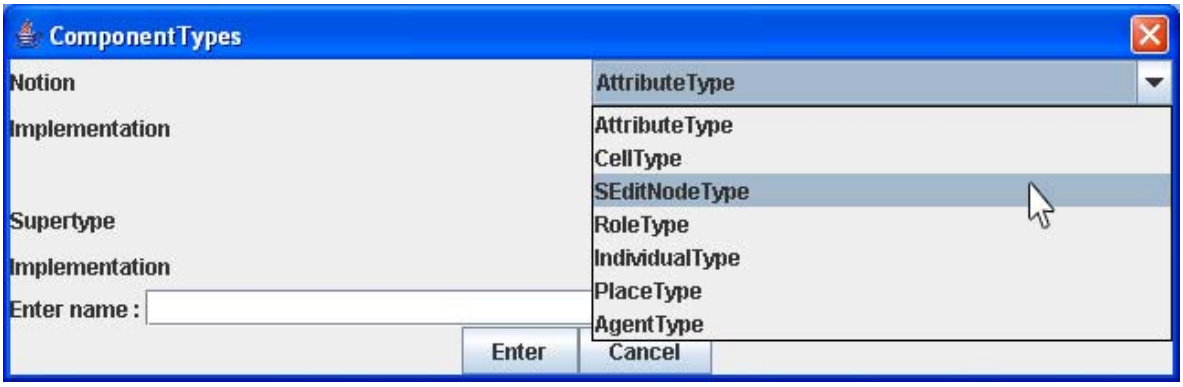



Transition valide

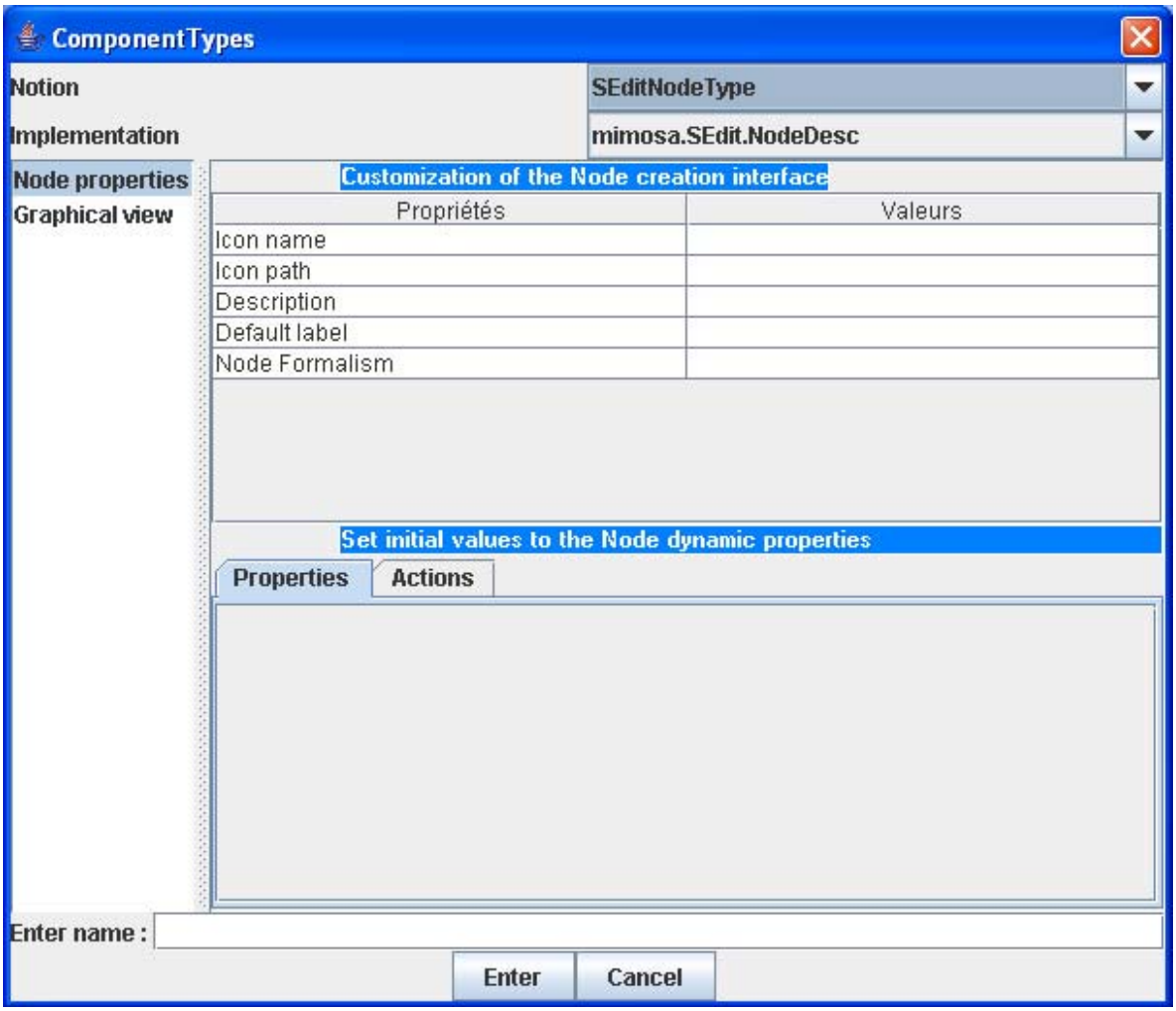
# B. Description des types de nœuds :

## 1. Description d'une place.

Ajouter un composant de la notion **SEditNodeType** :




L'écran suivant apparaît

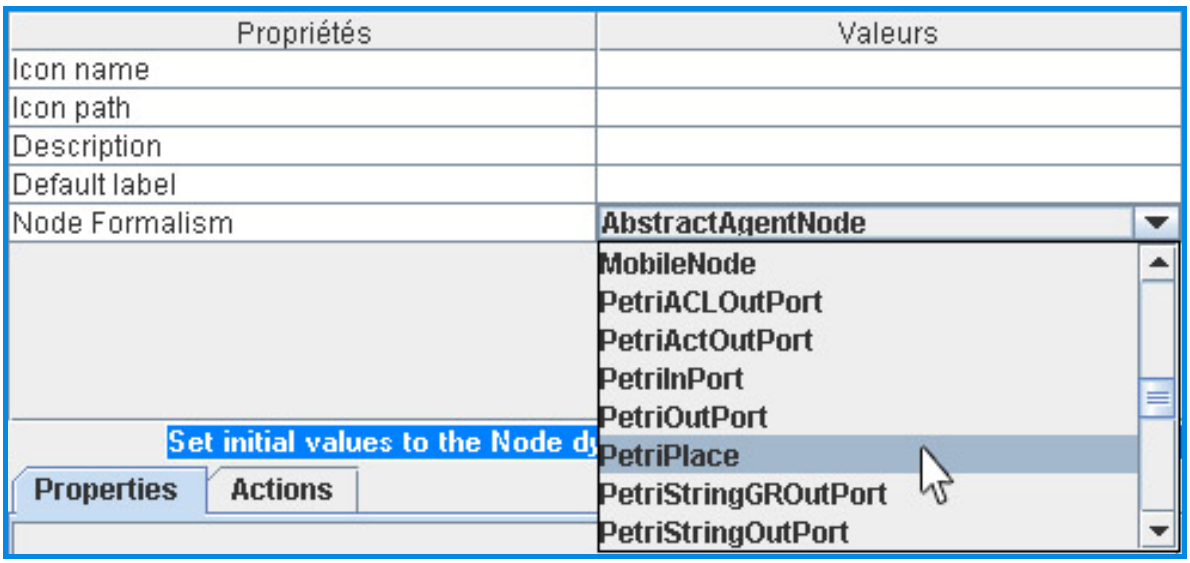


Attribuez une icône à la place pétri. Pendant la construction du modèle vous pourrez instancier une place en cliquant sur cette icône dans la barre d'outils.

| Propriétés | Valeurs |
|---|---|
| Icon name | |
| Icon path | /C:/Documents and Settings/Administrateu... |
| Description | |
| Default label | |
| Node Formalism | mimosa.SEdit.Formalisms.Petri.PetriPlace |

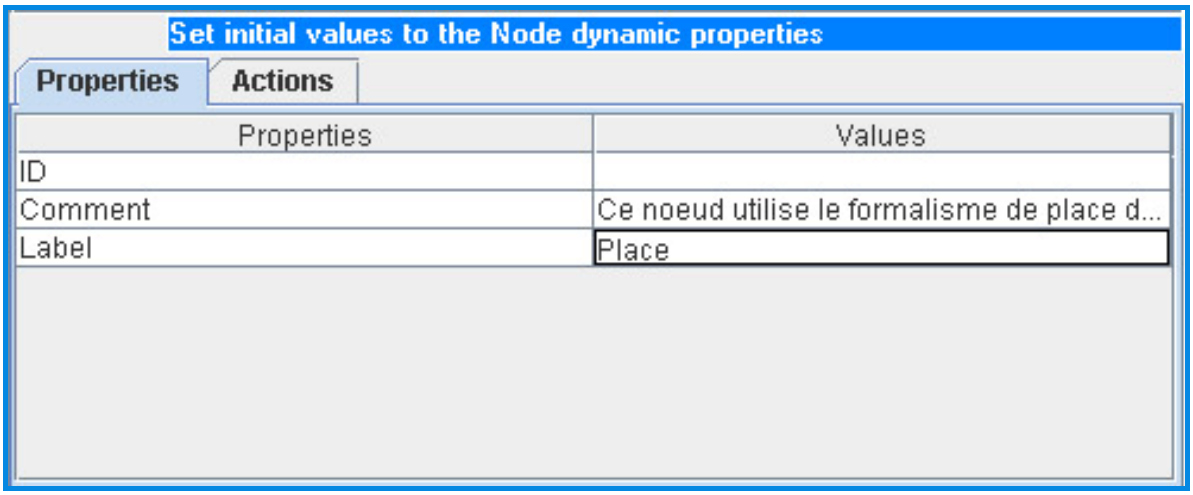
Dans cet exemple nous avons sélectionné l'icône suivante :

Sélectionnez le formalisme utilisé pour décrire le nœud, en l'occurrence le formalisme de place.




Donnez des informations complémentaires tels que la description qui devra être affiché, le label par défaut, etc.

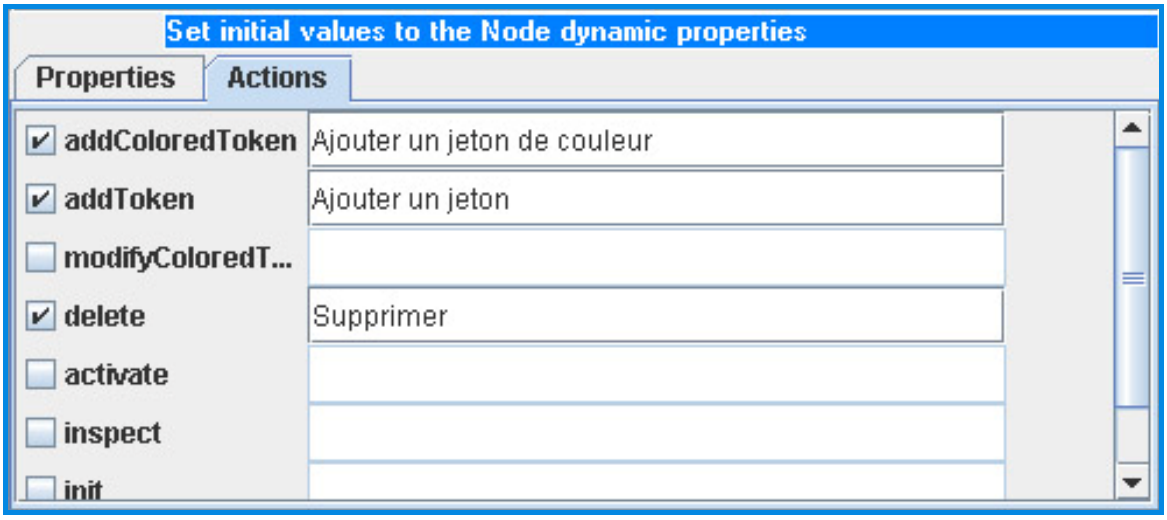
Attribuez des valeurs initiales par défaut à la place :



**Remarque :**
Le formalisme de la place pétri n'utilise que les propriétés de base du nœud. Par contre il définit plusieurs actions possibles.

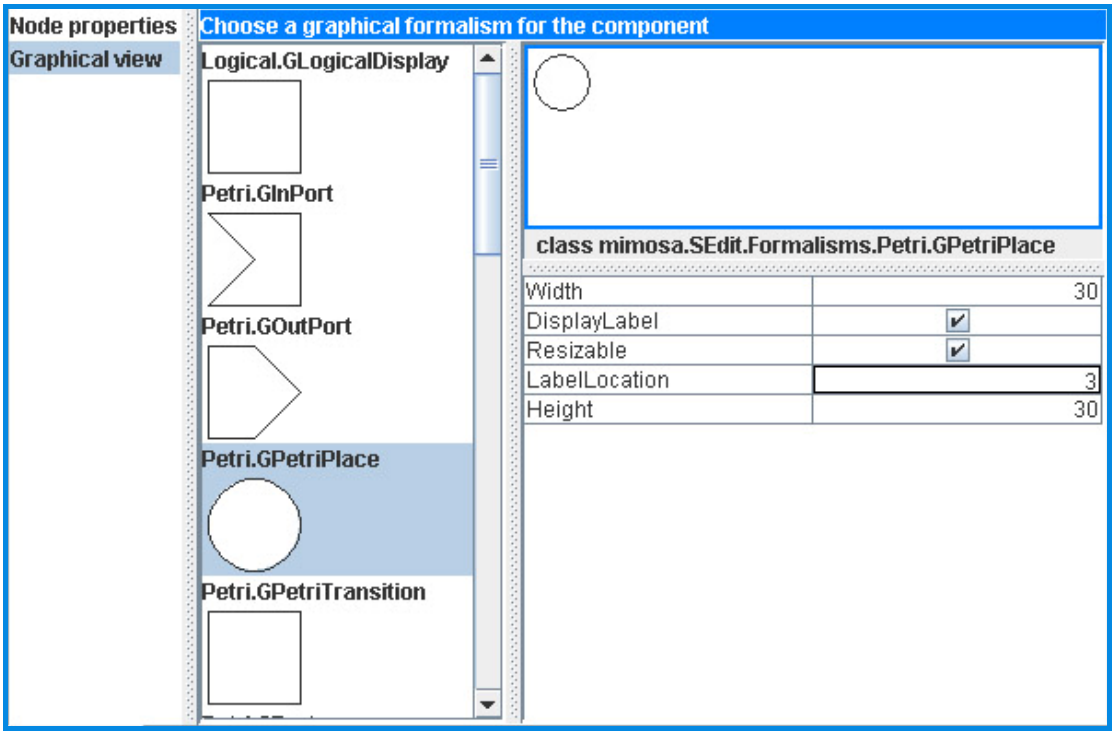
Choisissez à présent les actions qu'il sera possible d'exécuter sur la place Pétri.

Nous sélectionnerons les actions *ajouter un jeton*, *ajouter un jeton de couleur* et *supprimer*.



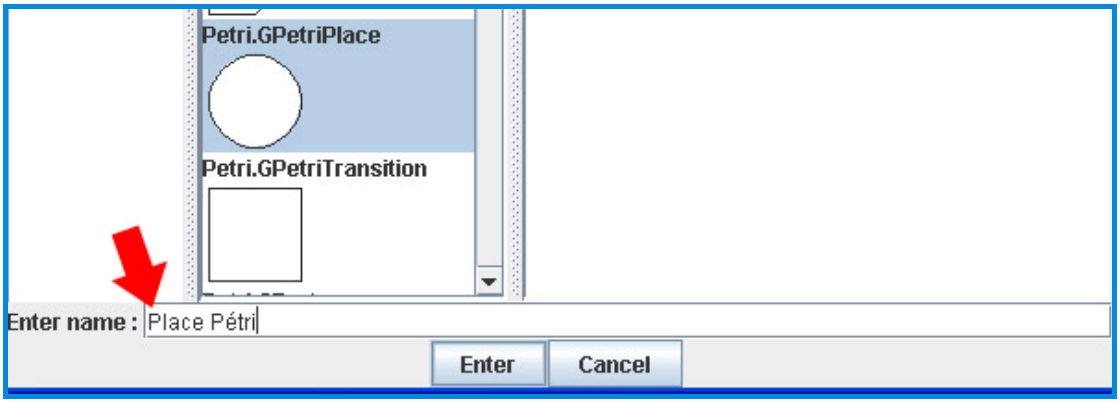

Nous venons de décrire un type de nœud : une place.
Il faut maintenant associer un formalisme graphique à la place Pétri.

Placez vous dans le panneau Graphical View et sélectionnez le formalisme graphique **GPetriPlace** :

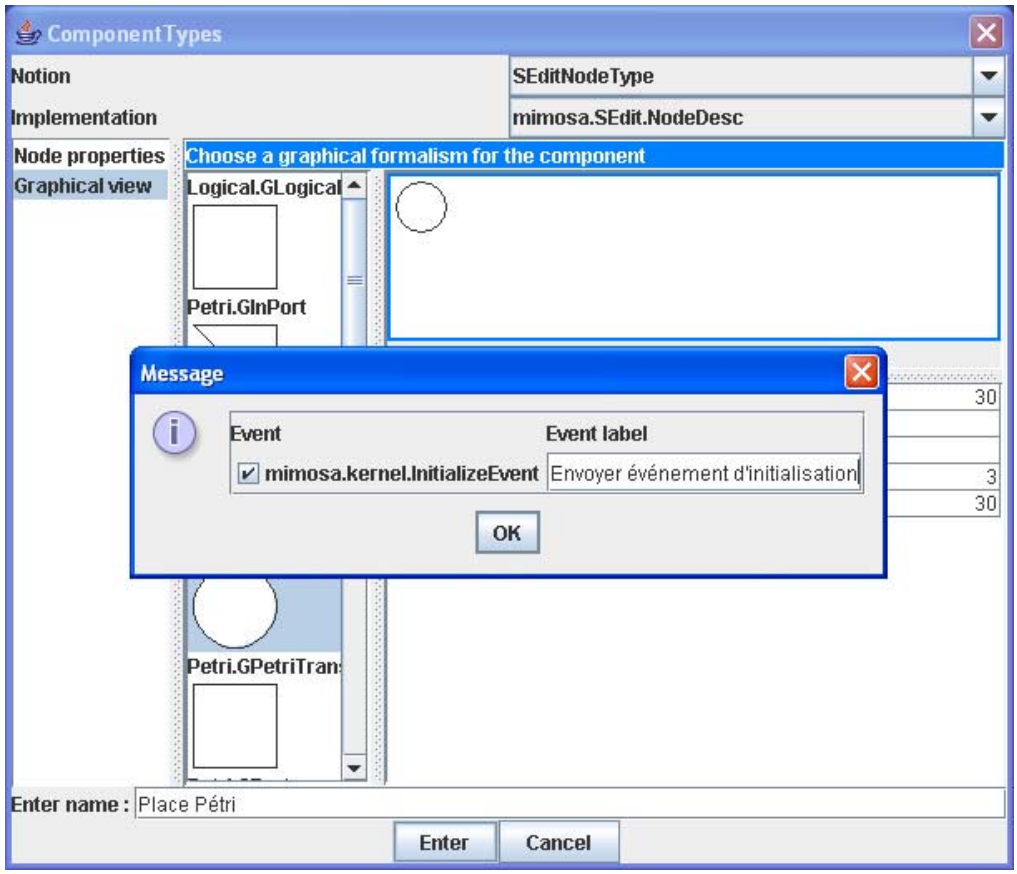



Définissez des valeurs initiales par défaut aux propriétés de l'objet graphique telles que la largeur et la hauteur.

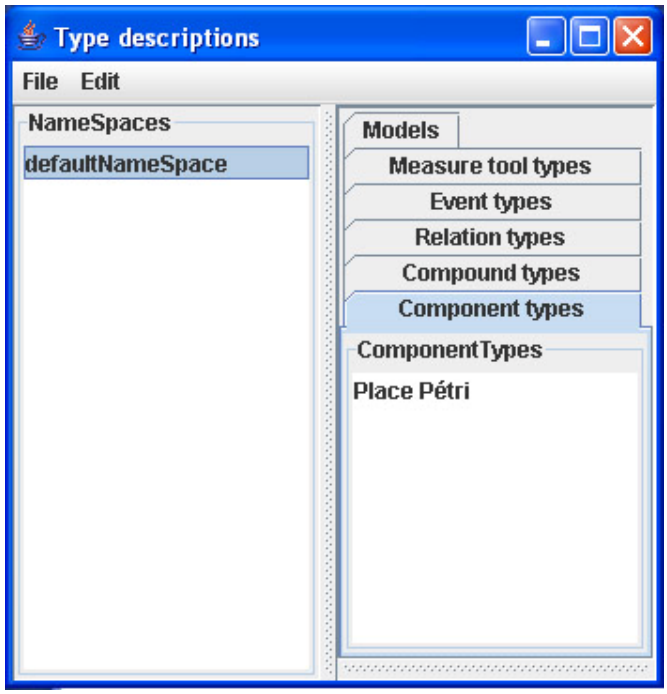
Attribuez un nom au type que vous venez de décrire :



Après validation, une boite de dialogue apparaît. Cette boite de dialogue permet de sélectionner les événements que le composants peut recevoir pendant l'édition des modèles dans la fenêtre SEdit.




Le type que nous venons de décrire apparaît à présent dans la liste des types de composants.

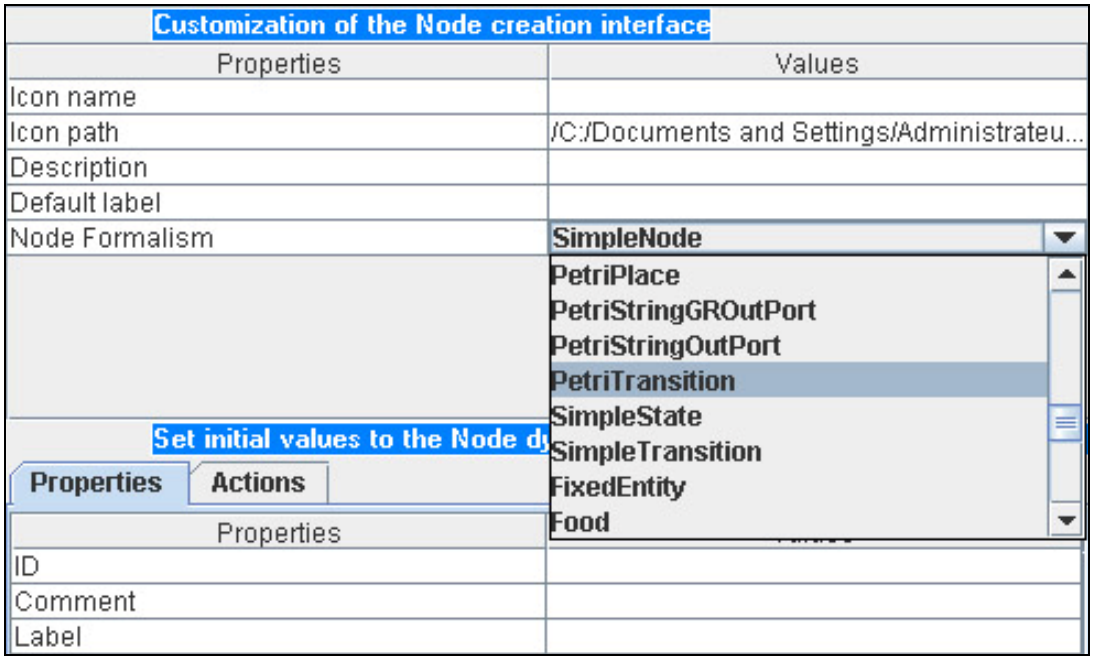


## 2. Description d'une transition

En suivant la même démarche, nous allons décrire une transition.

On associe la transition à l'icône : 

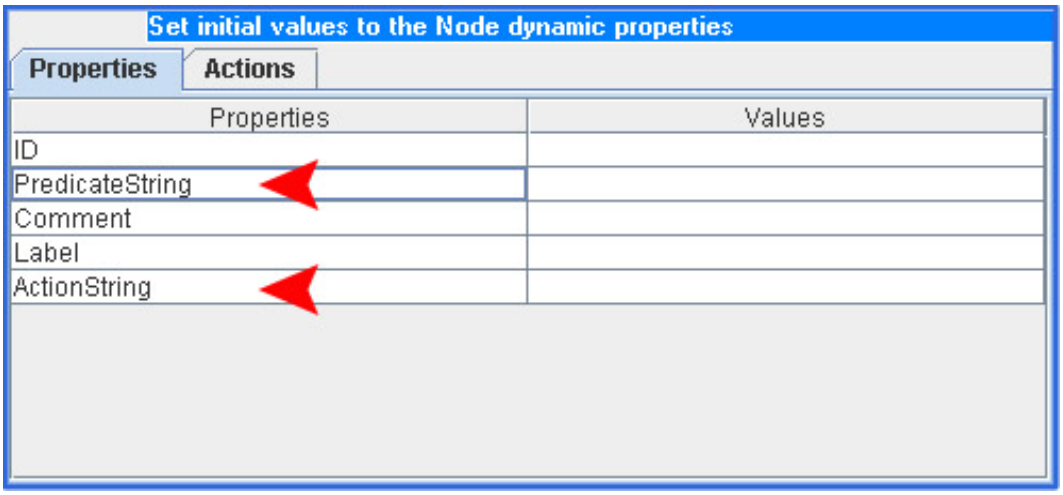
Pour modéliser une transition, le nœud doit être exprimé dans le formalisme décrit par la classe **PetriTransition** :




Une fois que le formalisme **PetriTransition** a été sélectionné vous pouvez attribuer des valeurs initiales à la transition. Ainsi le formalisme de la transition Pétri définit les attributs de nœuds suivants :

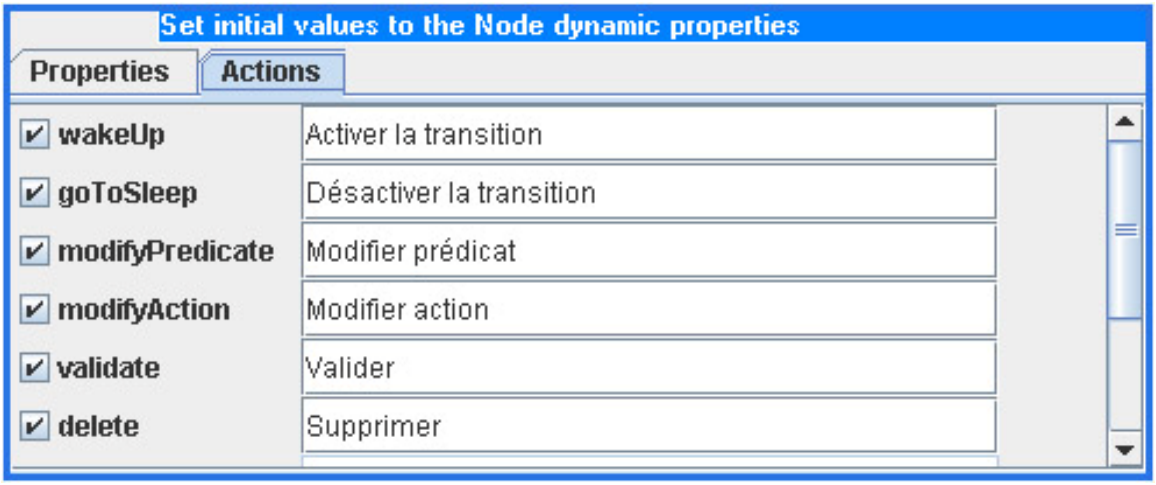
- Expression de prédicats (**PredicateString**) : elle constitue une condition de validation d'une transition.
- Expression d'actions (**ActionString**) : elle permet de définir l'action à exécuter lorsqu'une transition est activée.

Dans cet exemple nous ne fixerons cependant pas de conditions initiales à la validation d'une transition.

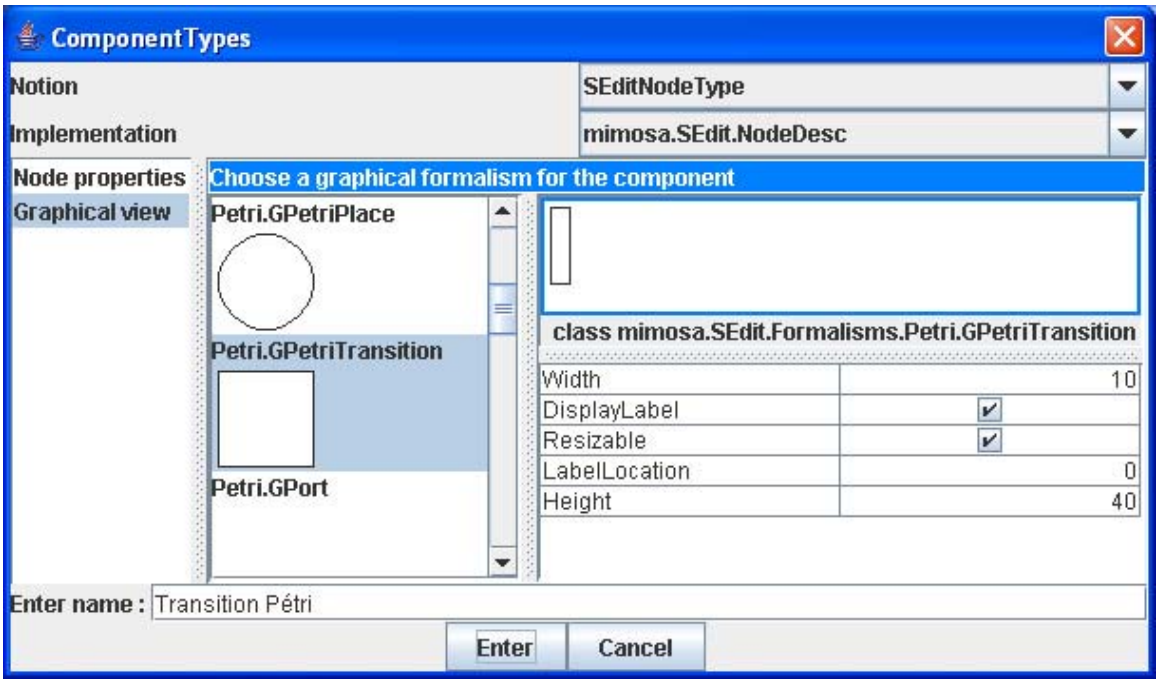


Il est possible d'exécuter les actions suivantes sur une transition :

- Activer transition (**WakeUp**) : rends actif une transition qui avait été désactivé.
- Désactiver transition (**GoToSleep**) : désactive une transition.
- Modifier le prédicat (**ModifyPredicate**) : modifie le prédicat d'une transition.
- Modifier l'action (**ModifyAction**) : modifie l'action d'une transition
- Valider (**Validate**) : permet de franchir une transition.

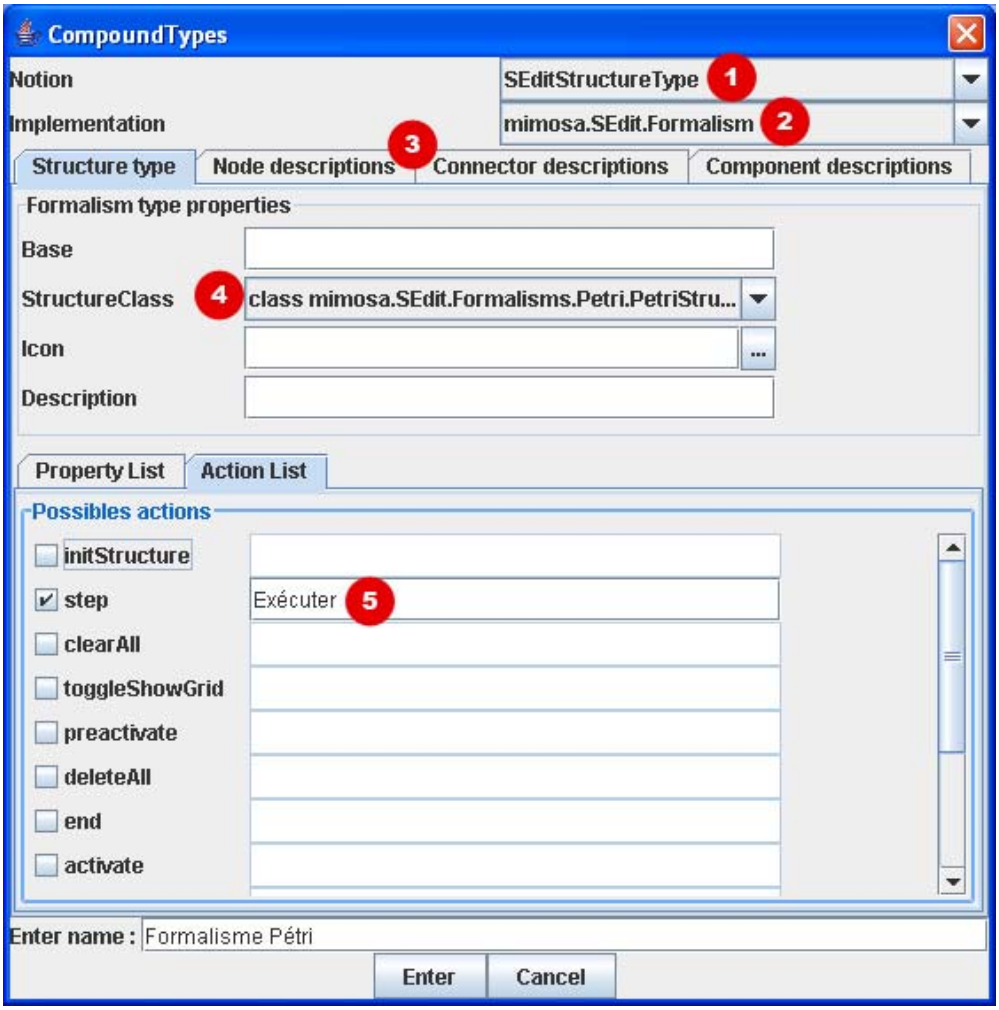



Associons à présent un formalisme graphique à la Transition Pétri : La classe **GPetriTransition** a été développé pour visualiser une transition pétri.



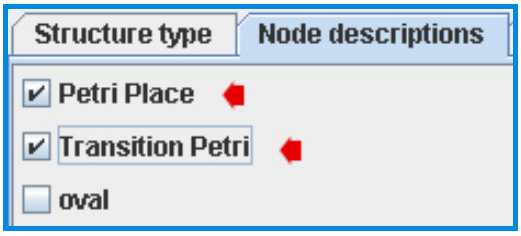

Le type que nous venons de définir sera appelé Transition Pétri.

## C. Description d'une structure Pétri.




La structure est un composé, son type est une instance de la classe **mimosa.SEdit.Formalism** (2). Les formalismes des types de structures sont catégorisés par la notion **SEditStructureType** (1).

Une structure est un composé, une agrégation de nœuds. Décrire une structure revient dans un premier temps à sélectionner les catégories de nœuds (3) qui pourront être instanciées dans la structure.




Sélections des types de noeuds

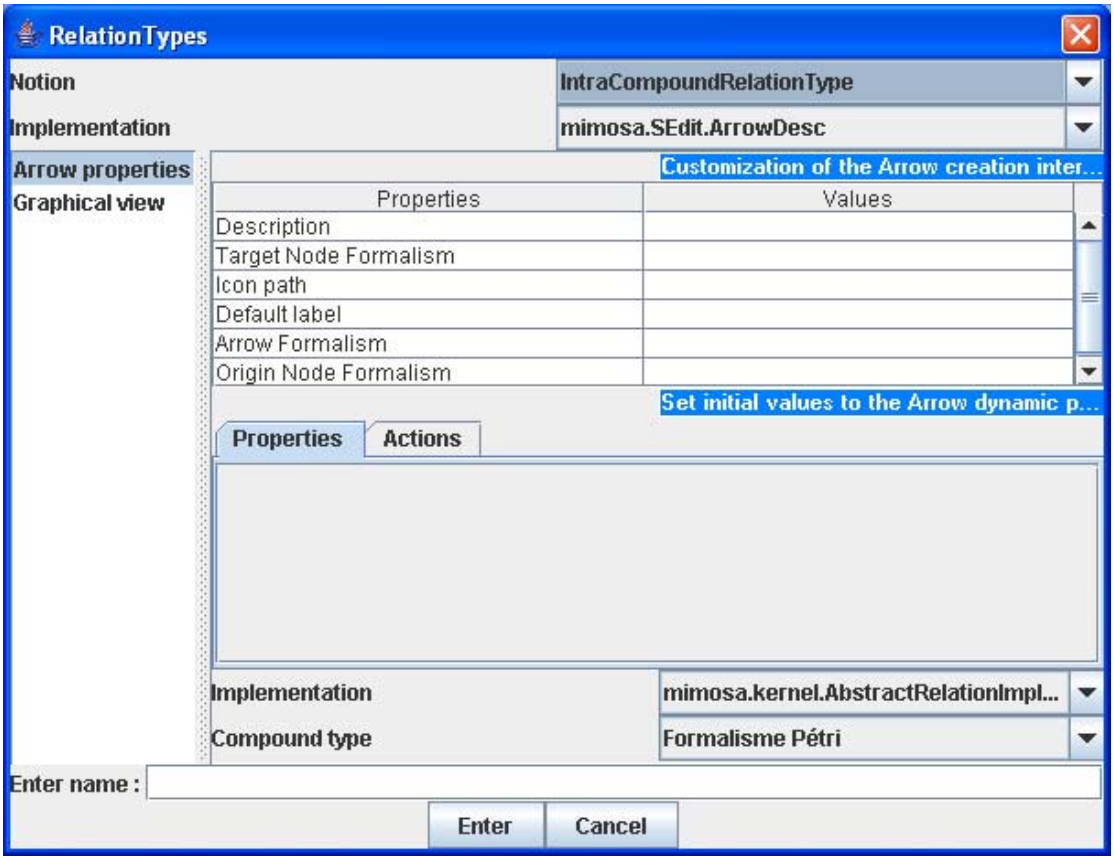
La notion de structure est implémentée dans plusieurs package de SEdit pour rendre compte de formalismes spécifiques. Dans le formalisme Pétri la structure est déclinée à travers la classe **mimosa.SEdit.Formalisms.Petri.PetriStructure** (4).

Dans le formalisme Pétri la structure ne se caractérise que par sa dynamique. En effet une structure dans le formalisme des réseaux de pétri doit disposer d'une action globale (5) de franchissement des transitions (**step**, label : exécuter).

# D. Description des types d'arcs :

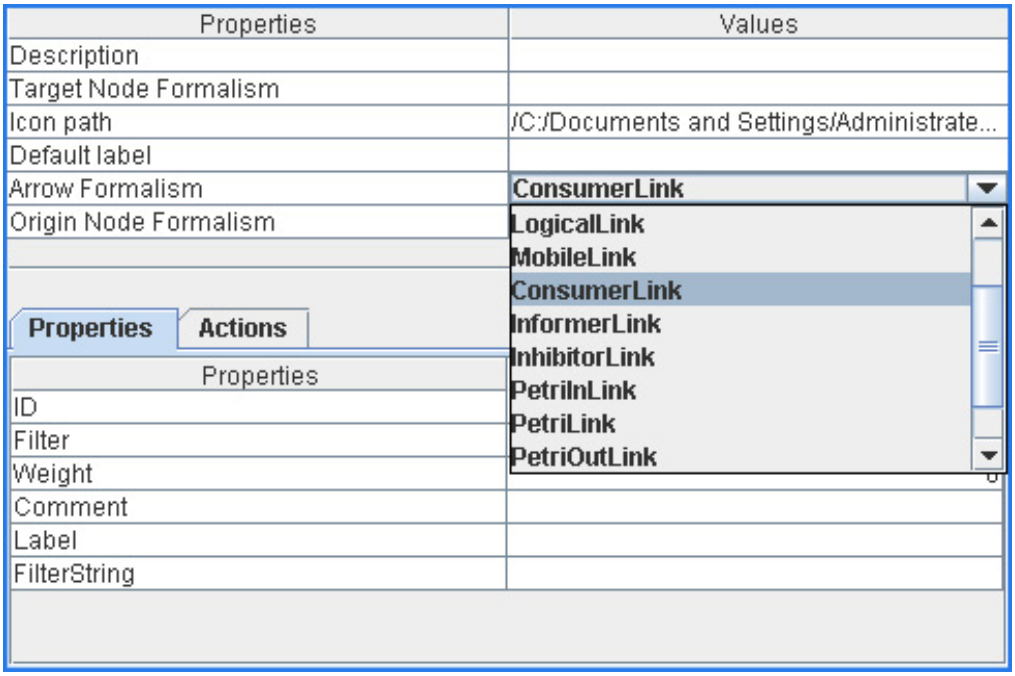
## 1. Description d'un arc entrant

Un arc entrant est un type de relation entre une place et une transition. Nous allons par conséquent décrire un type de relation intra composé implémenté par la classe **ArrowDesc**.



Nous attribuons l'icône de création suivante à l'arc :

L'attribut **Arrow Formalism** permet de sélectionner le formalisme des arcs. La classe **ConsumerLink** implémente la notion d'arc entrant.

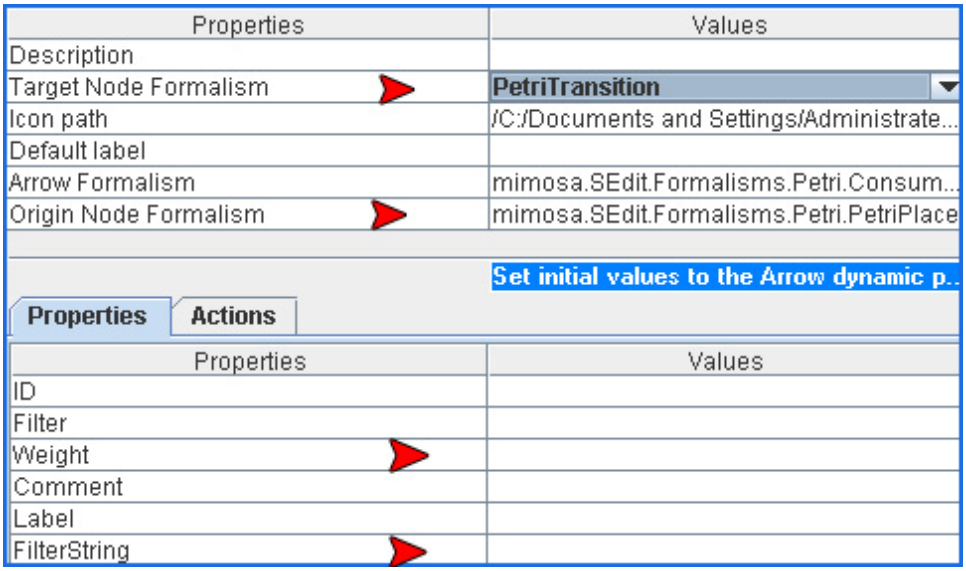



Le formalisme **ConsumerLink** définit les attributs d'arc suivants :

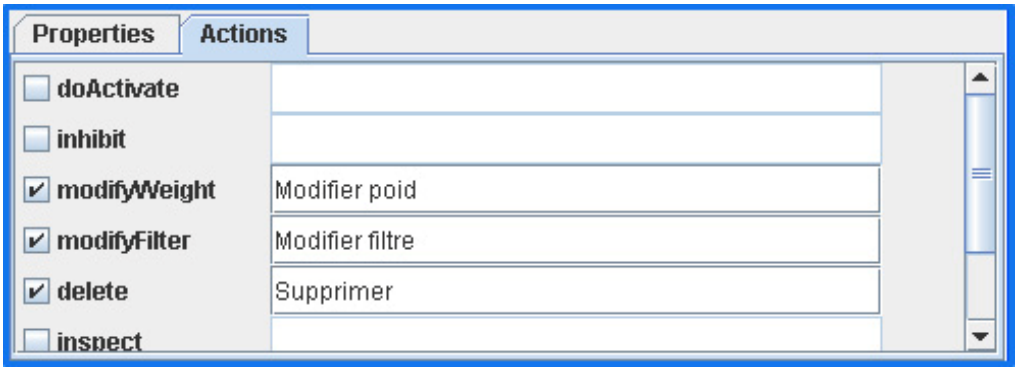
**Weight :** Le poids de la liaison en production et en consommation.
**FilterString :** Le filtre de la liaison.

Il s'agit de la définition d'un arc entrant entre une place et une transition, il nous faut par conséquent spécifier le formalisme du nœud initial et le formalisme du nœud terminal.

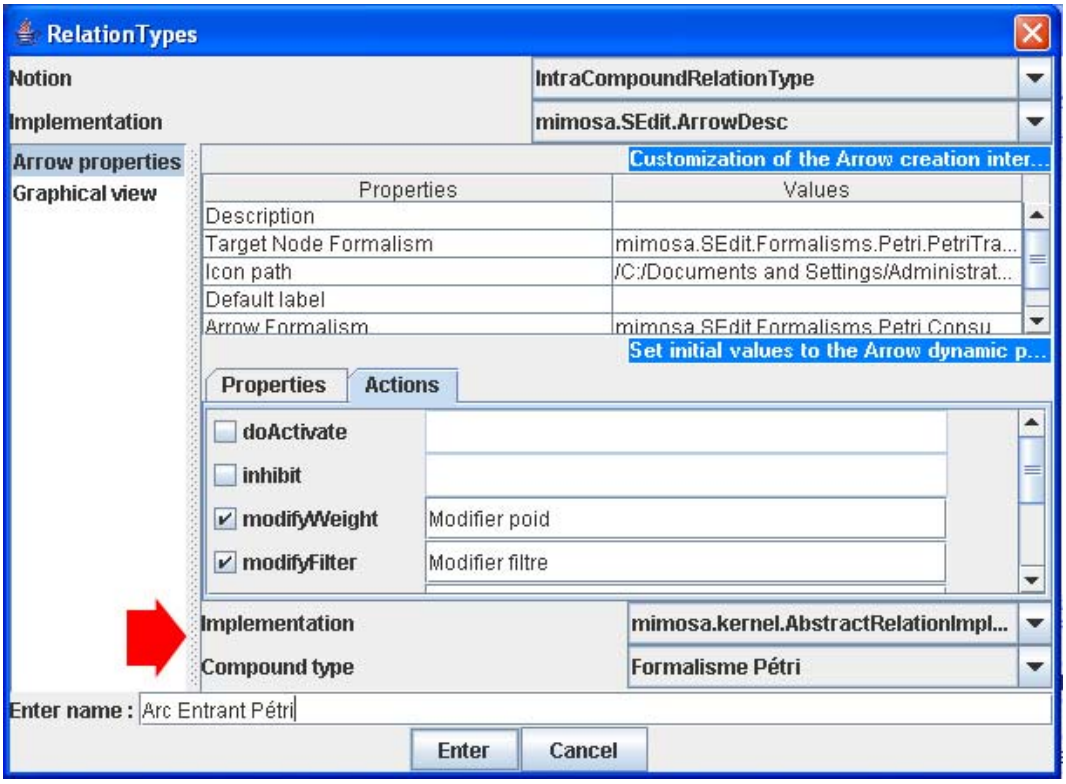
Le nœud initial d'un arc entrant est une place, l'attribut **Origin Node Formalism** aura pour valeur : **PetriPlace**.
Le nœud terminal d'un arc entrant est une transition, l'attribut **Target Node Formalism** aura pour valeur : **PetriTransition**.



Parmi les actions possible sur un arc entrant nous choisirons les actions **ModifyWeight**, **ModifyFilter**, **Delete** auxquelles nous attribuerons les noms respectifs : *Modifier poids*, *Modifier filtre*, *Supprimer.*

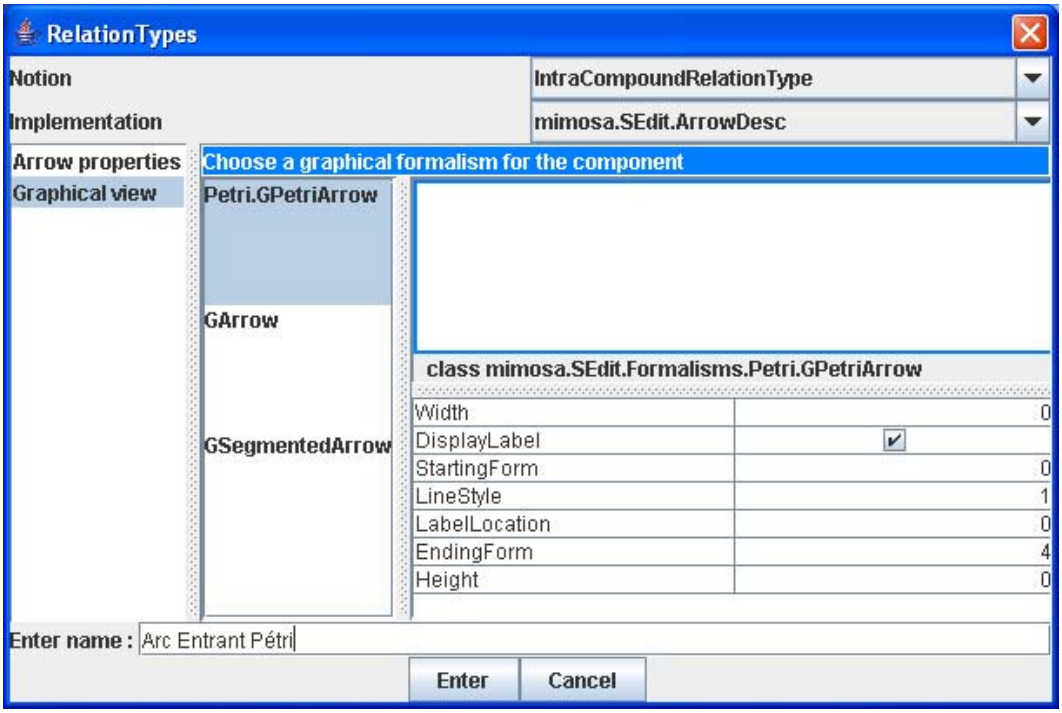



Ensuite il nous faut définir le composé auquel appartient la relation, nous choisirons le composé **Formalisme Pétri** que nous avions décris plus haut.



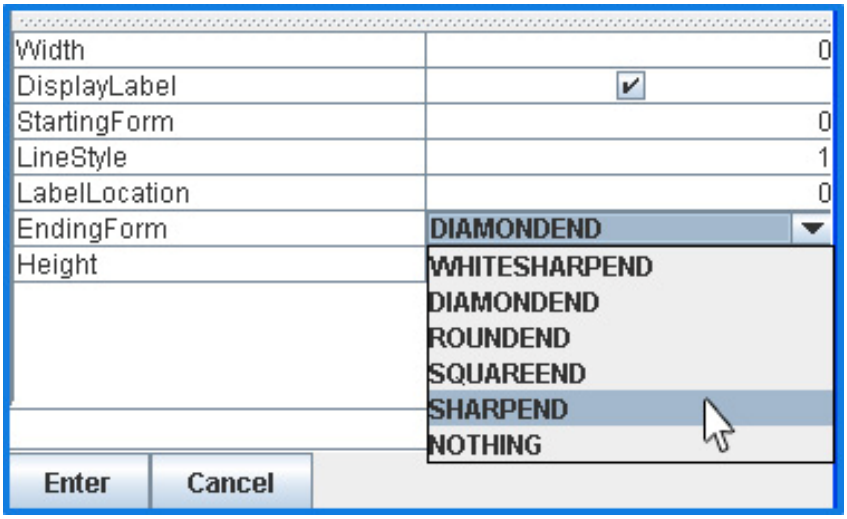
## Formalisme graphique associé

Le choix de la notion graphique à associer au type de relation se fait dans le panneau **Graphical view**.

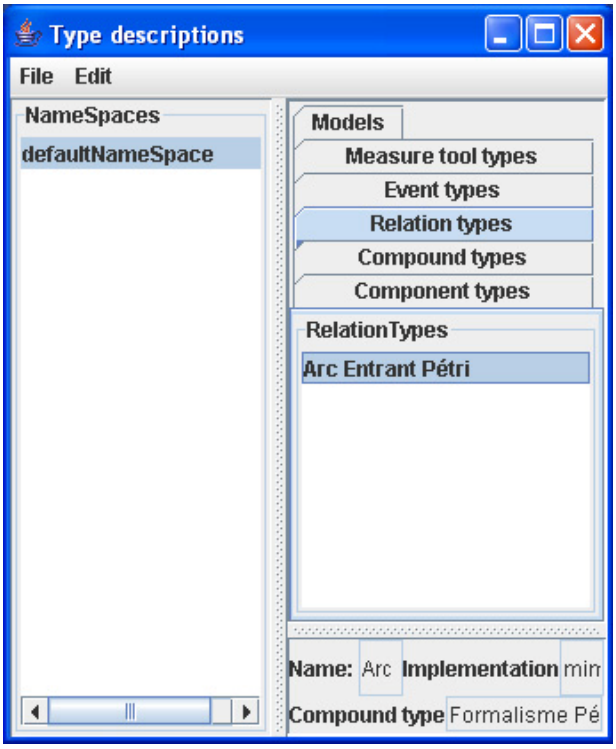



La notion graphique d'arc entrant est implémentée par la classe **GPetriArrow**.

Il nous faut définir les attributs **LineStyle** à DIRECTLINE et **EndingForm** à SHARPEND. Ce pour spécifier qu'il s'agit d'une flèche directe avec un bout en triangle standard.



Nous appellerons ce type de relation : **Arc Entrant Pétri**. Il apparaît dans la liste des relations comme l'illustre la figure suivante.

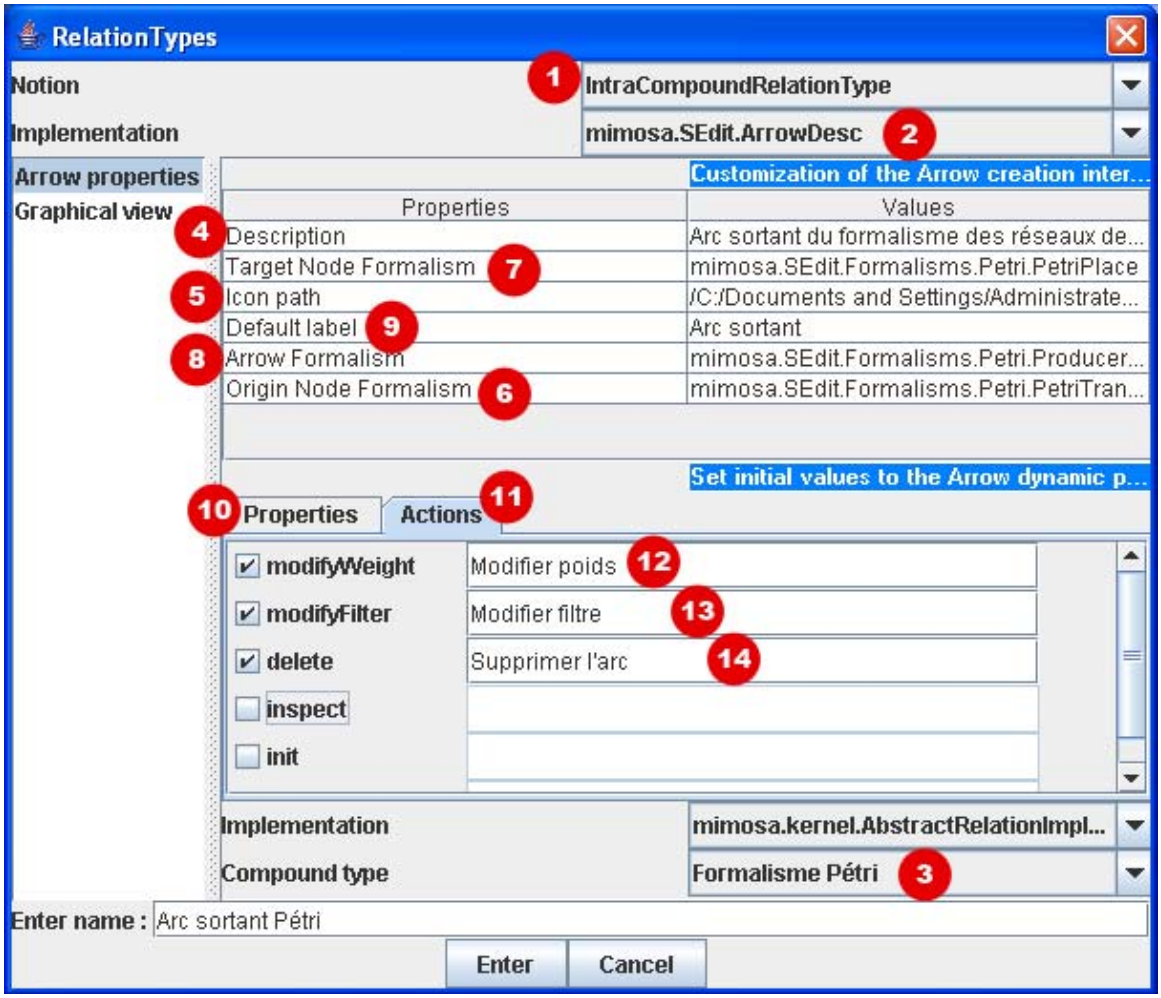


## 2. Description d'un arc sortant

**Remarque :** Souvent, nous utilisons le terme formalisme pour désigner la classe d'implémentation d'une notion.

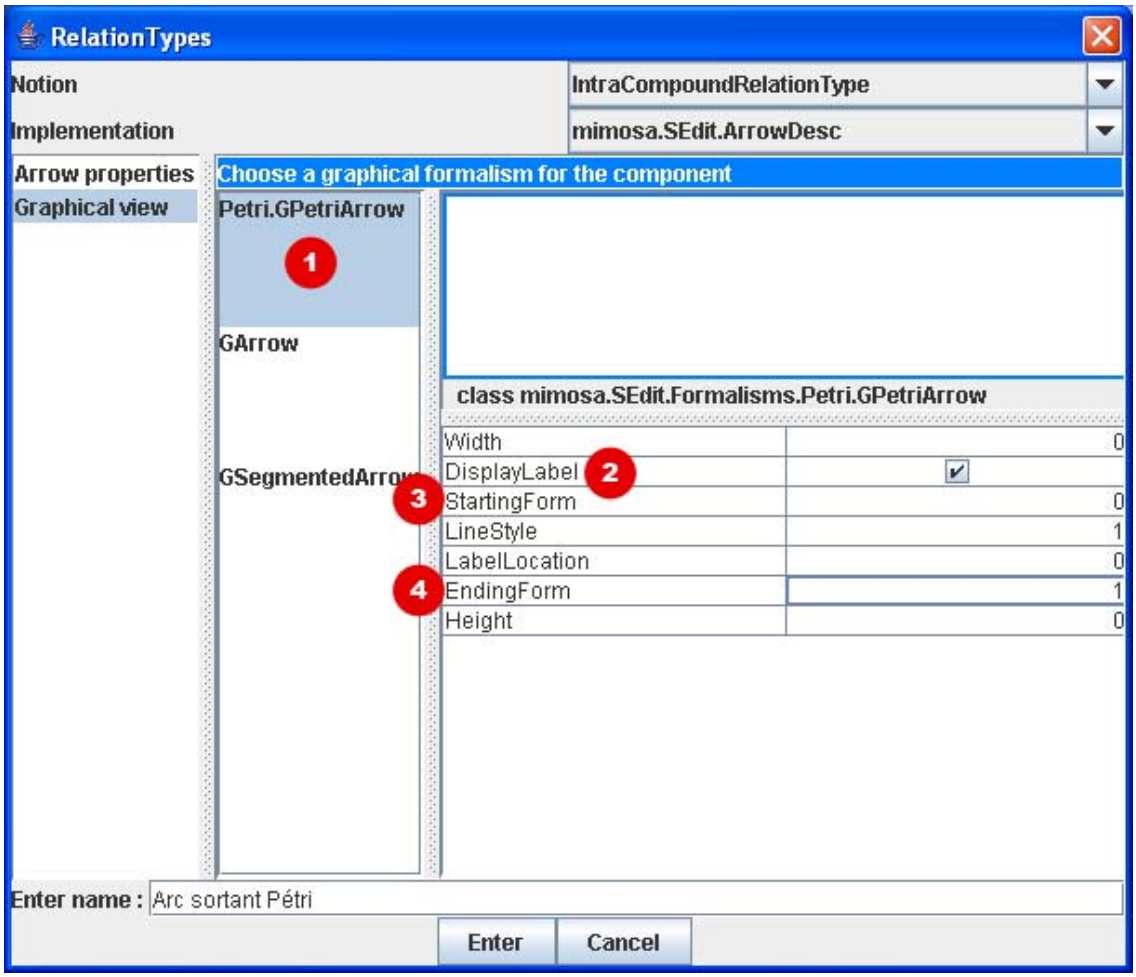
| i. DESCRIPTION DU TYPE | | |
|---|---|---|
| 1 | Notion | IntraCompoundRelationType |
| 2 | Classe d'implementation | Mimosa.SEdit.ArrowDesc |
| **1. Attributs de la notion** | | |
| 3 | Composé cible de la relation | Formalisme Pétri |
| 4 | Description de l'arc | Arc sortant du formalisme des réseaux de Pétri |
| 5 | Icône associée à la notion |  Producer.gif |
| 6 | Formalisme du nœud initial de l'arc | mimosa.SEdit.Formalisms.Petri.PetriTransition |
| 7 | Formalisme du nœud terminal de l'arc | mimosa.SEdit.Formalisms.Petri.PetriPlace |
| 8 | Formalisme de l'arc | Mimosa.SEdit.Formalisms.Petri.PetriProducerLink |
| 9 | Label par défaut du type d'arc | Arc sortant |
| 10 | Attribution de valeurs initiales aux attributs de l'instance du type en cours de définition | |
| Aucune attribution nécessaire | | |
| 11 | Sélection des actions possibles | |
| 12 | Action 1 | Action : **modifyWeight**, Label : **Modifier poids** |
| 13 | Action 2 | Action : **modifyFilter**, Label : Modifier filtre |
| 14 | Action 3 | Action : **delete**, Label : Supprimer arc |

## Capture d'écran :



| ii. DESCRIPTION DU TYPE : Forme graphique associé | | |
|---|---|---|
| 1 | Formalisme | mimosa.SEdit.Formalisms.Petri.GPetriArrow |
| **1. Description du type graphique** | | |
| 2 | Affichage du label | activé |
| 3 | Forme au début du tracé de l'arc | Aucune forme au départ de la ligne (Valeur : **NOTHING**) |
| 4 | Forme à la fin du tracé de l'arc | Forme fléchée à la fin de l'arc (Valeur : **SHARPEND**) |

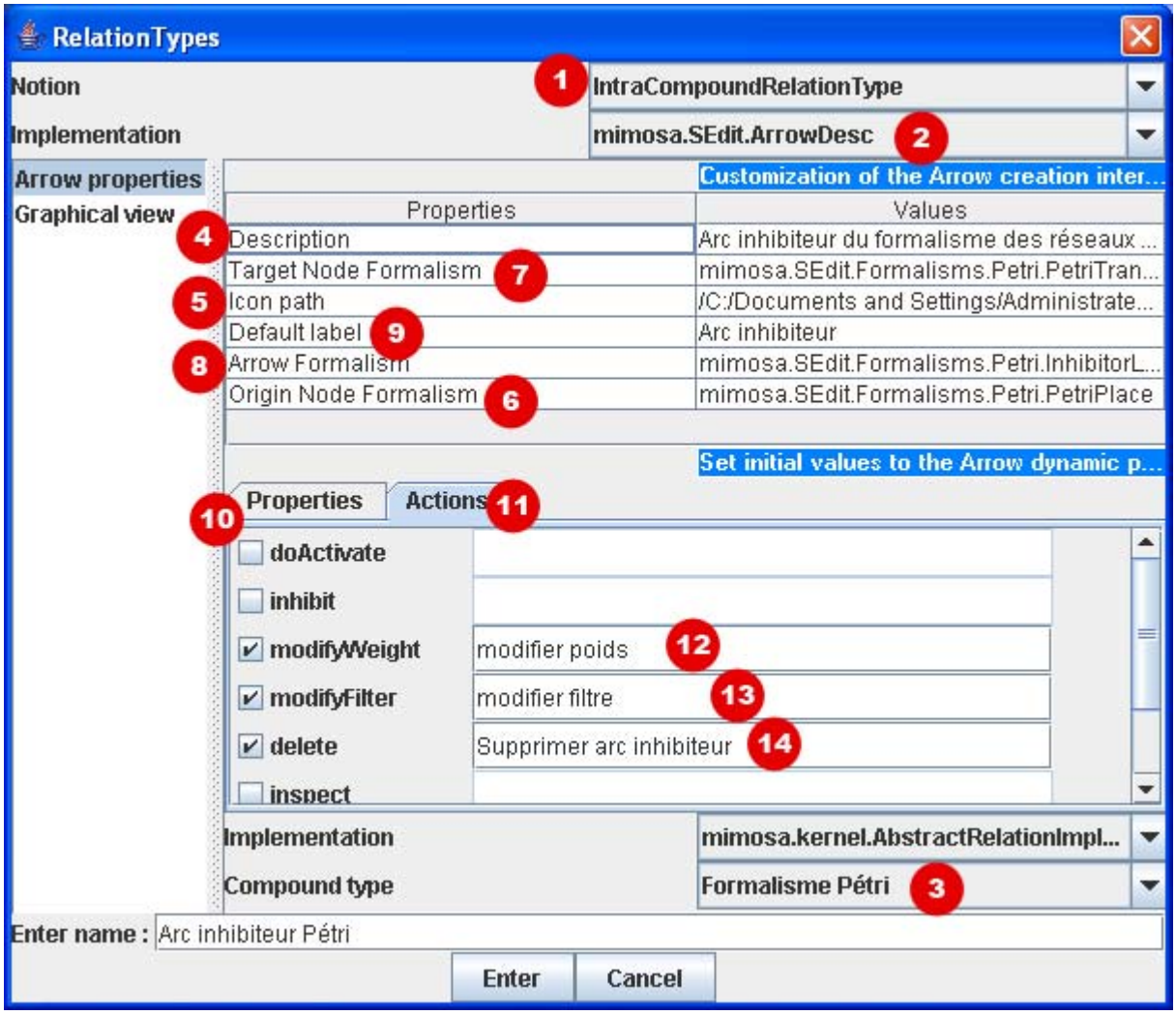
## Capture d'écran :



## 3. Description d'un arc inhibiteur

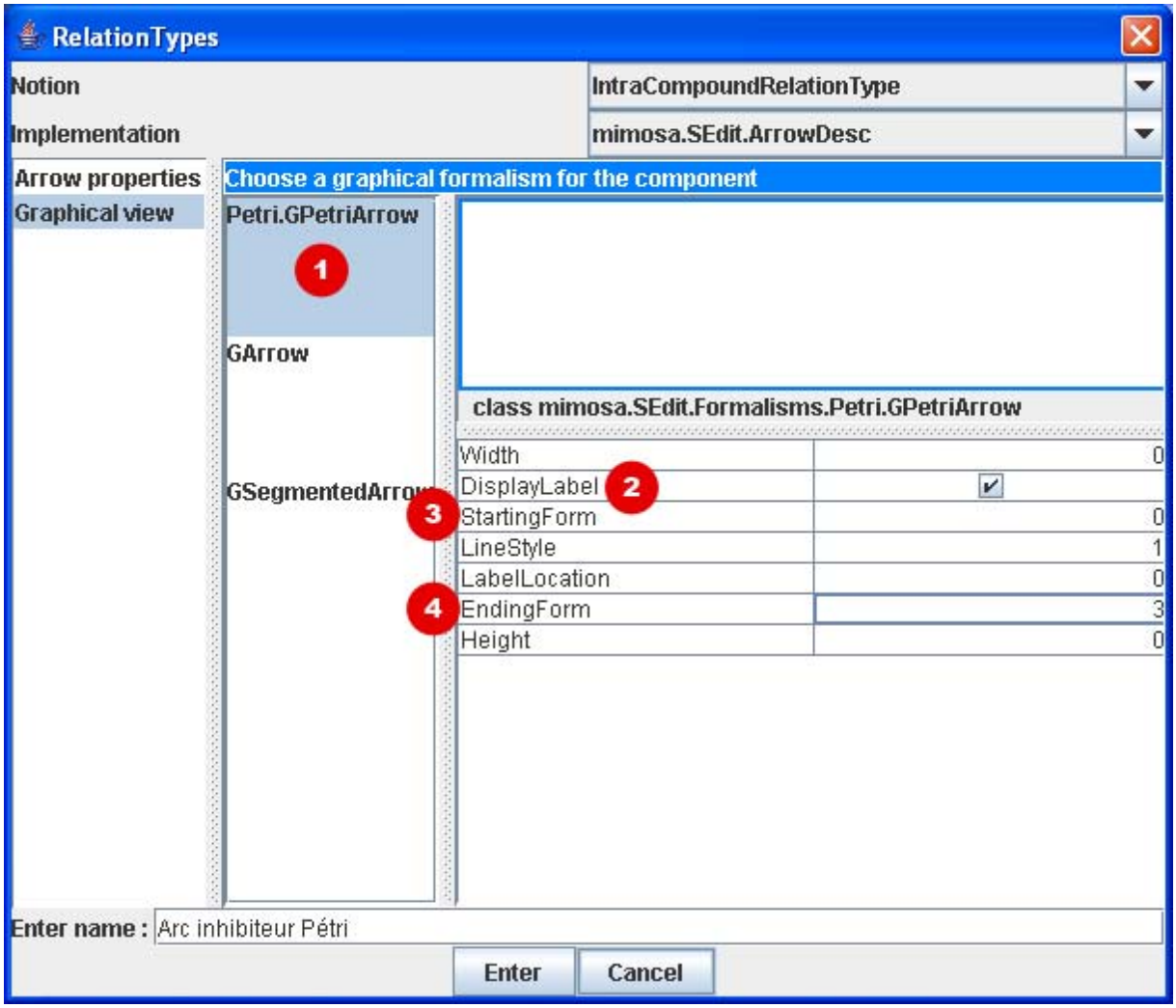
| ii. DESCRIPTION DU TYPE | | |
|---|---|---|
| 1 | Notion | IntraCompoundRelationType |
| 2 | Classe d'implementation | Mimosa.SEdit.ArrowDesc |
| **1. Attributs de la notion** | | |
| 3 | Composé cible de la relation | Formalisme Pétri |
| 4 | Description de l'arc | Arc inhibiteur du formalisme des réseaux de Pétri |
| 5 | Icône associée à la notion |  inhibitor.gif |
| 6 | Formalisme du nœud initial de l'arc | mimosa.SEdit.Formalisms.Petri.PetriPlace |
| 7 | Formalisme du nœud terminal de l'arc | mimosa.SEdit.Formalisms.Petri.PetriTransition |
| 8 | Formalisme de l'arc | Mimosa.SEdit.Formalisms.Petri.PetriInhibitorLink |
| 9 | Label par défaut du type d'arc | Arc inhibiteur |
| 10 | Attribution de valeurs initiales aux attributs de l'instance du type en cours de définition | |
| Aucune attribution nécessaire | | |
| 11 | Sélection des actions possibles | |
| 12 | Action 1 | Action : **modifyWeight**, Label : **Modifier poids** |
| 13 | Action 2 | Action : **modifyFilter**, Label : Modifier filtre |
| 14 | Action 3 | Action : **delete**, Label : Supprimer arc |

## Capture d'écran :



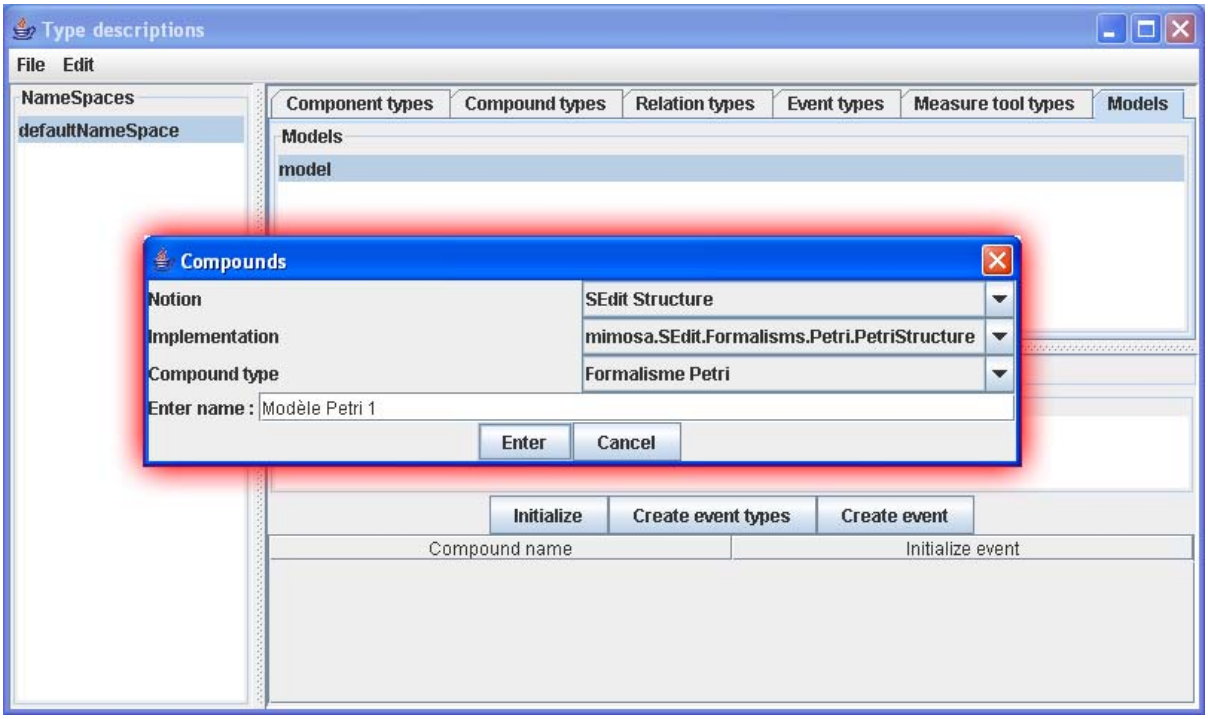
| v. DESCRIPTION DU TYPE : Forme graphique associé | | |
|---|---|---|
| 1 | Formalisme | mimosa.SEdit.Formalisms.Petri.GPetriArrow |
| **1. Description du type graphique** | | |
| 2 | Affichage du label | activé |
| 3 | Forme au début du tracé de l'arc | Aucune forme au départ de la ligne (Valeur : **NOTHING**) |
| 4 | Forme à la fin du tracé de l'arc | Forme fléchée à la fin de l'arc (Valeur : **SHARPEND**) |

## Capture d'écran :

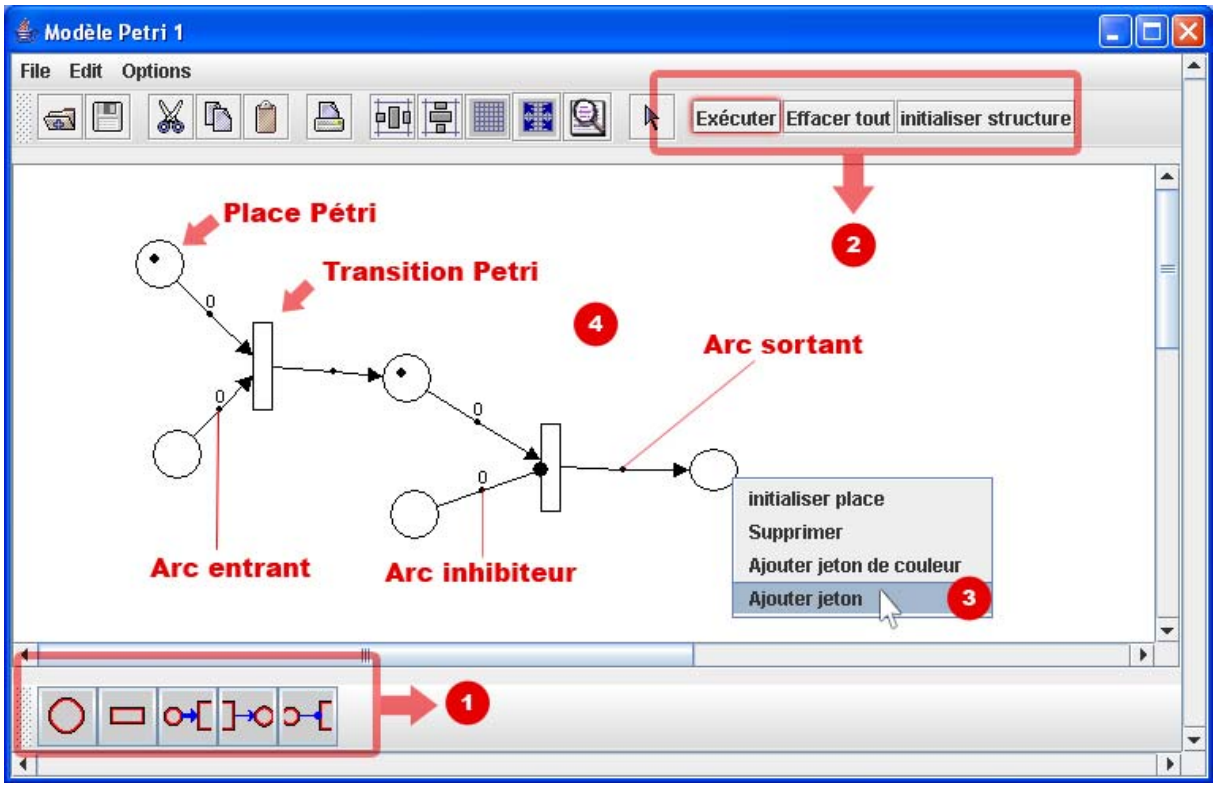


# E. Edition d'un modèle Petri

Un modèle Pétri dans MIMOSA est un composé, un point de vue d'un modèle multi formalisme. Pour instancier un modèle Pétri, il faut donc instancier une notion de composé SEdit (**SEdit Structure)** implémenté par la classe **mimosa.SEdit.Formalism.Petri.PetriStructure**. Cette classe définit le formalisme pour le modèle, en tant que structure, des réseaux de Pétri.




Le modèle Pétri est un composé du modèle, sa fenêtre de visualisation est la fenêtre S-Edit d'édition de structure de diagrammes.



Les types définis précédemment apparaissent dans la barre d'outils de SEdit (1). Il suffit d'actionner le bouton associé au type pour pouvoir positionner des instances du type dans l'espace d'édition (4). Dans le cas d'une relation, d'un arc, l'instanciation consiste à la mise en relation de deux nœuds.

L'action globale de simulation définie dans le type de structure apparaît au niveau de la barre d'outil du coin supérieur gauche de l'écran (2).

Les actions ainsi que les événements définis sur les nœuds et les arcs sont accessibles sur clic droit de la souris, ainsi l'action **Ajouter jeton** (3) d'une place ajoute un jeton à la place.

Si la transition est validée (coloration fushia), l'action **Valider** a pour effet de provoquer le franchissement de la transition.